ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Grado en Ingeniería Informática,

Mención computación

Sistema de recomendación basado en redes neuronales competitivas

Recommendation system based on competitive neural networks

Realizado por
Teresa Rocha Muñoz
Tutorizado por
Juan Miguel Ortiz de Lazcano Lobato
Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, NOVIEMBRE 2016

Fecha defensa:
El Secretario del Tribunal

Resumen: En la presente memoria se expone la implementación del algoritmo de un sistema de recomendación basado en el PCACL (Principal Components Analysis Competitive Learning), que es una red neuronal competitiva que realiza un análisis de componentes principales (PCA) en cada neurona. Además, se exponen las razones por las cuales se ha elegido este algoritmo como el núcleo del sistema de recomendación y la importancia de que se realice un buen entrenamiento de la red neuronal.
La implementación del sistema de recomendación se ha realizado en el lenguaje de programación R con la ayuda del entorno de trabajo RStudio, de los cuáles se detallan las características más relevantes en el proceso de esta implementación.
Finalmente se realizan y estudian las pruebas realizadas al sistema para comprobar si es fiable o no y si se podría mejorar el algoritmo.

Palabras claves: Sistema de recomendación, redes neuronales competitivas, clasificación de productos, lenguaje de programación R.

Abstract: In the present specification is exposed the recommendation system algorithm implementation based in PCACL (Principal Components Analysis Competitive Learning), it is a neural network competitive that make a principal components analysis (PCA) in each neuron. Also, it is exposed the reason why this algorithm has been chosen as the core of this recommendation system and the importance of a good training of the neural network.

The implementation of the recommendation system has been performed in the programming language R with the help of the RStudio work environment, which details the most relevant characteristics in the implementation process.

Finally, the tests performed on the system are carried out and analyzed to check whether it is reliable or not.

Keywords: Recommendation system, competitive neural networks, product classification, programming language R

Índice

Introducción

En este trabajo se presenta la realización y puesta en marcha de un sistema de recomendación basado en redes neuronales artificiales siguiendo un nuevo modelo que combina el aprendizaje competitivo clásico con un análisis de componentes principales (Principal Components Analysis, PCA) en cada neurona. Este nuevo modelo se denomina en inglés "*Principal Components Analysis Compeitive* Learning" (PCACL). (López Rubio, Ortiz de Lazcano Lobato, Muñoz Pérez, & Gómez Ruiz, 2004)

Las redes neuronales artificiales, también denominadas redes neuronales o simplemente redes, son sistemas de aprendizaje y procesamiento automático basados en el comportamiento de las redes neuronales biológicas. Estos sistemas están compuestos por unidades de procesamiento, denominadas neuronas, que interactúan entre sí para producir una salida.

El análisis de componentes principales, en adelante PCA, es una técnica estadística que reduce la dimensión de un conjunto de datos encontrando la proyección con la que mejor se representan los datos y manteniendo la mayor cantidad de información posible. (Análisis de componentes principales, s.f.)

Por otra parte, los sistemas de recomendación son sistemas de filtrado de información que buscan predecir la preferencia que un determinado usuario tendría hacia un producto dado, de manera que sólo se filtren, y no se le presenten, elementos que el sistema considera que no le interesarán al usuario en cuestión. (Sistema de recomendación, s.f.)

Hoy en día es posible encontrar sistemas de recomendación integrados en una gran cantidad de aplicaciones como sistemas de ventas, sistemas de información (sobre películas, libros, artículos de investigación, etc.) e, incluso, redes sociales, donde a los usuarios se les recomiendan productos u otros elementos marcados como preferidos por algunos de los usuarios conectados directamente con él (usuarios amigos).

Entre los diversos enfoques a seguir a la hora de diseñar un sistema de recomendación algunos de los más utilizados han sido el del vecino más cercano (Nearest Neighborhood), que basa su recomendación en los vecinos más cercanos (Sistema de recomendación, s.f.), y el algoritmo de filtrado colaborativo, que basa su recomendación de productos en la información que proporcionan usuarios con gustos parecidos. Para tal fin, los usuarios suelen dividirse en grupos, compuestos por aquellos usuarios con una preferencia similar hacia un determinado grupo de productos. Finalmente, la recomendación, de un determinado producto a un usuario dado, se haría utilizando la información grupal. (Filtrado colaborativo, s.f.)

Es muy normal que un sistema tenga registrados una amplia variedad de productos, pero los usuarios no suelen calificar o comprar más que una pequeña

cantidad de ellos por lo que resulta complicado encontrar usuarios en los que basar nuestra recomendación dado un usuario concreto. Para solventar este problema se ha pensado en utilizar técnicas de reducción de la dimensión, de forma que el espacio de entrada sea más compacto y los grupos que defina el sistema neuronal presenten una entropía[1] menor y sean más significativos.

El modelo de red neuronal PCACL es una red no supervisada del tipo competitivo. Se ha optado por realizar el sistema de recomendación basado en una red PCACL porque, este tipo de redes, están diseñadas especialmente para cuantificar el espacio de entrada, es decir, dividen los distintos patrones de entrada en grupos de elementos con cierta semejanza entre ellos. Así se solventa uno de los mayores problemas a los que se enfrentan los sistemas de recomendación, que es la dispersión de los datos con los que se trabaja.

El modelo PCACL representa una mejora respecto a los métodos conocidos de PCA ya que no es necesario presentar el conjunto de datos completo a la red en cada paso de cómputo y mantiene las propiedades de reducción de la dimensión del PCA mejorando la velocidad de ejecución de la red. (Jolliffe, 1986)

Para implementar el sistema de recomendación se ha optado por el lenguaje de programación R puesto que permite trabajar con grandes volúmenes de datos y se distribuye bajo la licencia GNU GPL (Carmona, 2007). Cabe destacar que R es una implementación del lenguaje S y está enfocado hacia el análisis estadístico (R Development Core Team, 2000).

Por último, mencionar que la presente memoria se divide en siete capítulos indicados a continuación.

- En el primer capítulo se hace referencia a la formación adquirida sobre el modelo PCACL y el lenguaje R, así como a introducir al lector en las redes neuronales artificiales, en el análisis de componentes principales y en el modelo PCACL.
- En el segundo capítulo se exponen los pasos que se realizan en el programa principal del sistema de recomendación, entre otros los que se han dado para generar los datos de entrada finales del sistema de recomendación, las acciones para implementar el procesamiento de los datos y la configuración e inicialización de la red, así como su entrenamiento y posterior clasificación de los datos.
- En el tercer capítulo se detallan las pruebas realizadas al sistema para verificar su correcto funcionamiento, así como los datos generados para dichas pruebas.

[1] La entropía de la información o un conjunto de datos es la cuantificación de la incertidumbre de la fuente de esa información o de los datos

- En el cuarto capítulo se menciona el diseño e implementación del sistema de recomendación para su uso.

Tecnologías a utilizar

La implementación del modelo PCACL se ha realizado en el lenguaje de programación R (versión 3.3.1) con la ayuda del entorno de trabajo RStudio (versión 0.99.896).

Se ha optado por el lenguaje de programación R ya que es un potente lenguaje de análisis estadístico y ofrece grandes funcionalidades para la tarea que se lleva a cabo en este trabajo. A pesar de ser ya un lenguaje muy potente, a día de hoy, sigue en continuo crecimiento y, como se ha mencionado anteriormente, es de software libre (R (lenguaje de programación), s.f.). R también proporciona un entorno de desarrollo propio, aun así, se ha utilizado el entorno de RStudio que ofrece muchas funcionalidades más amigables que las del entorno R.

El lenguaje R permite definir scripts en el que se definen funciones, que al ser ejecutados declaran la función en el entorno de variables globales para que sea accesible durante todo el proceso de trabajo.

El entorno RStudio facilita la creación de un proyecto y guarda automáticamente todos los datos que han intervenido en la última sesión que realizamos del proyecto. Así, cada vez que iniciamos el proyecto, éste mantiene el entorno global y el historial de todo lo que se ha realizado en dicho proyecto, es decir, no es necesario leer los scripts que contienen las funciones cada vez que iniciemos RStudio. Esto es una gran ventaja respecto a versiones anteriores de RStudio en las que había que guardar manualmente el entorno de variable globales o los datos que se deseaban mantener durante la ejecución.

Como se verá más adelante todo esto facilita mucho la implementación del sistema de recomendación puesto que se ha creado un proyecto en R denominado *"Sistema de recomendación"* que está compuesto por un script principal y otros en los que se han definido funciones para ayudar en la inicialización, el entrenamiento, las pruebas y las recomendaciones del sistema.

Capítulo I. Documentación y formación

Se ha realizado un estudio de los diferentes sistemas de recomendación actuales y el modelo PCACL, en el cual se basa el sistema de recomendación desarrollado en esta memoria. Además, se han ampliado los conocimientos sobre el lenguaje R y sobre RStudio.

Redes neuronales artificiales

Las redes neuronales artificiales son modelos matemáticos basados en el comportamiento de las redes neuronales biológicas y están compuestas por unidades de proceso denominadas neuronas. Las neuronas producen una salida para un patrón dado, así como las neuronas del sistema neuronal biológico producen una respuesta a un estímulo dado. La red está compuesta por capas, en las que se producen las interconexiones de las entradas y las neuronas, dichas conexiones, denominadas sinápticas, tienen asignado un peso y ayudan en el aprendizaje de la red. El aprendizaje, de las redes neuronales, es el proceso de presentar los patrones a la red y cambiar los pesos de las conexiones sinápticas según una regla de aprendizaje. (Ballesteros, s.f.)

Las redes neuronales se pueden clasificar según su estructura (topología) o según su manera de aprender.

Según la topología, las redes neuronales pueden ser monocapa (ver figura 1) o multicapas (ver figura 2). En cambio, según la forma de aprender de la red pueden ser supervisadas y no supervisadas.

Las redes de aprendizaje supervisado son aquellas en las que, dados unos patrones de entrada, se conoce la salida que debe dar la red y por lo tanto los pesos se modifican para ajustar la entrada a dicha salida. En cambio, las redes no supervisadas sólo presentan los patrones de entrada y la red clasificará los patrones en función de las características que tengan en común.

Cabe destacar que existe otro tipo de red en el que se combina el aprendizaje supervisado y el no supervisado, en el cuál a la red no se le presentan los patrones objetivos, pero en el aprendizaje se le indica si las salidas son acertadas o no. (Ballesteros, s.f.)

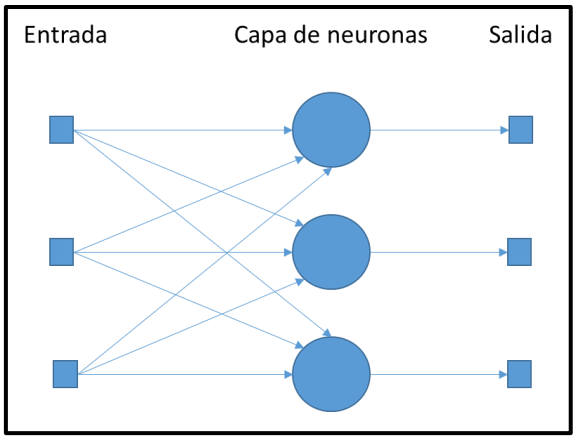


Figura 1. Ejemplo de red neuronal monocapa

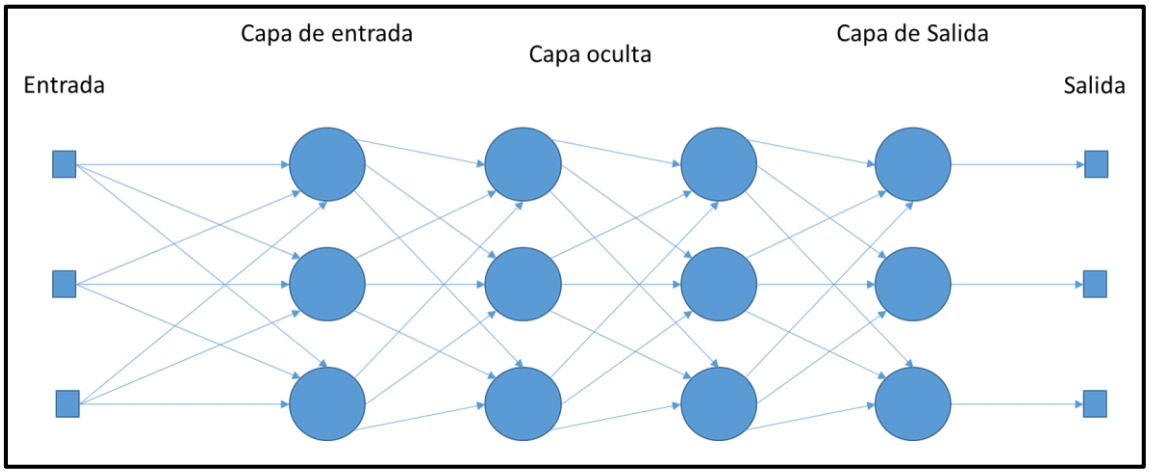


Figura 2. Ejemplo de red neuronal multicapa

Hay muchos tipos de redes neuronales, pero, para comprender este trabajo, interesa centrarse en las redes neuronales competitivas por lo que se va a ver brevemente la diferencia entre las redes competitivas y las colaborativas.

Las redes neuronales colaborativas, como su nombre indica, son redes en las que las neuronas colaboran para representar los patrones de entrada, en cambio, las redes neuronales competitivas compiten por ser la neurona que representa mejor al patrón de entrada y la ganadora se llevará todo el aprendizaje de ese patrón. Así, el objetivo de las redes neuronales competitivas es que se formen grupos de patrones que son representados por una única neurona cada uno. (Ballesteros, s.f.)

Análisis de componentes principales (PCA)

El análisis de componentes principales, comúnmente conocido por sus siglas en inglés PCA, es una técnica utilizada para reducir la dimensionalidad de un conjunto de datos, es decir, identifica el número más pequeño de variables sin

correlación que explican la máxima cantidad de varianza. Este conjunto de variables resultantes se denominan componentes principales. (¿Qué es el análisis de componentes principales?, s.f.)

El PCA realiza una transformación lineal que escoge un nuevo sistema de coordenadas para el conjunto original de datos en el cual la varianza de mayor tamaño del conjunto de datos es capturada en el primer eje y será el primer componente principal, la segunda varianza más grande será el segundo eje y así sucesivamente hasta tener todos los componentes principales del conjunto de datos. Para construir esta transformación lineal, se debe obtener la matriz de covarianza y los autovectores de la misma que constituirán la nueva base vectorial.

Modelo de red neuronal PCACL

El método PCACL, sobre el que se basa el sistema de recomendación, es una red competitiva no supervisada de una sola capa. Las redes competitivas tienen algunas limitaciones puesto que los modelos sólo proporcionan información sobre la media de cada grupo y esto conlleva una gran pérdida de información. En cambio, los métodos de análisis de componentes principales (PCA) no tienen este problema puesto que obtienen las direcciones principales de los datos, es decir, las direcciones de máxima varianza, así reducen la dimensión de los datos y almacenan la información más relevante. Además, se ha demostrado que PCA es una técnica lineal óptima para la reducción de la dimensión en referencia al error cuadrático medio (Jolliffe, 1986).

Por todo esto, la red neuronal PCACL presenta una mejora respecto a otros modelos neuronales puesto que, realiza un análisis de componentes principales (PCA) en cada neurona además de seguir un aprendizaje competitivo clásico y no es necesario presentar el conjunto de datos completo a la red en cada paso de cómputo, así se mantienen las propiedades de reducción de la dimensión del PCA y el cálculo se realiza de forma incremental.

El modelo PCACL tiene la capacidad de estimar la dimensión intrínseca de los datos, es decir, cada neurona es capaz de modificar su comportamiento para adaptarse a la dimensión local de la distribución de entrada.

Capítulo II. Implementación de la red neuronal

Como se ha mencionado anteriormente, se ha creado un proyecto, en RStudio, denominado *"Sistema de recomendación"*, el cual está compuesto por distintos scripts y funciones que realizan las diferentes tareas que componen el sistema de recomendación y que facilitan la utilización del algoritmo implementado para realizar diferentes pruebas o generar diferentes sistemas de recomendación, si así se quisiera. Con generar diferentes sistemas de recomendación se pretende indicar que se pueden entrenar diversas redes con distintos parámetros de aprendizaje, distinto número de neuronas (máximo número de grupos diferentes en los que se clasificarán los datos), las variables iniciales de estas neuronas serán diferentes también y se podrán entrenar con diferentes datos de entrenamiento y esto dará lugar a sistemas de recomendación adaptados a un objetivo concreto.

El script principal de este sistema se denomina *"Principal"* (ver ilustración 1) y se encarga de leer dos scripts principales, uno denominado *"Preparar entorno y datos"* y otro *"Ejecutar entrenamientos y predicciones"*. El primer script cargar las librerías necesarias, además de leer y guardar las funciones que se han definido para llevar a cabo la creación del sistema de recomendación, así como, ejecutar las instrucciones necesarias para generar los datos y preprocesarlos (ver ilustración 2). El segundo script, *"Ejecutar entrenamientos y predicciones"*, lleva a cabo la inicialización de la configuración y las variables de la red, además del entrenamiento y las pruebas y/o las predicciones según convenga (ver ilustración 3).

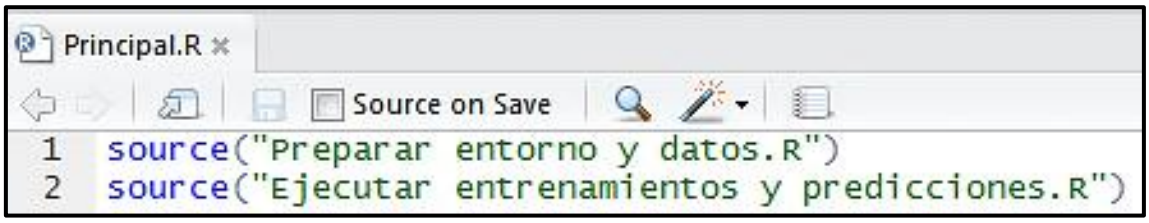


Ilustración 1. Principal

Preparar entorno y datos.R

Source on Save

```r
# CARGA DE LIBRERIAS #

library("matrixcalc", lib.loc="~/R/win-library/3.2")
library("Matrix", lib.loc="C:/Program Files/R/R-3.2.5/library")
library("ellipse", lib.loc="~/R/win-library/3.2")


# CARGA DE FUNCIONES #

source('Normalizar datos.R')
source('Inicializar variables de la RN.R')
source('Calcular bases vectoriales.R')
source('Entrenar RN.R')
source('Neurona ganadora.R')
source('Actualizar variables de la RN.R')
source('Clasificar datos.R')
source('Mostrar resultados.R')
source('Generar datos.R')

accion <- "P1"

switch(accion,
       R0={
          dim <- 1000
          tam <- 300
          datos <- generarDatosRecomendacion(dim, tam)
          dir <- "Datos del SR/"
          write.table( datos, paste(dir,"Datos.txt") )
       },
       P1={
          radio <- 10
          lim <- 5
          dim <- 10
          datos <- generarDatosArco(radio, lim, dim)
          write.table( datos, "Pruebas/Datos 1/Datos 1.txt" )
       },
       P2={
          rango <- 50
          multiplo <- 30
          datos <- generarDatosSeparados(rango, multiplo)
          write.table( datos, "Pruebas/Datos 2/Datos 2.txt" )
       },
       stop("No se ha encontrado la acci?n deseada")
)


# PREPROCESAMIENTO DE LOS DATOS #

datos_normalizados <- normalizar(datos) # se normalizan los datos

num_total_datos <- ncol(datos) # numero total de datos
dim_datos <- nrow(datos) # dimension de los datos
```

Ilustración 2 Preparación del entorno y de los datos

En la ilustración anterior se puede observar que al comenzar se cargan las librerías y se leen los scripts que contienen la definición de las funciones

necesarias para llevar a cabo las pruebas y generar el sistema de recomendación.

Cabe destacar que, para realizar una acción u otra, es decir, iniciar el sistema de recomendación o de las pruebas, es necesario dar un valor válido a la variable acción. Estos valores se pueden observar en la instrucción *"switch"* y son "R0" para generar los datos del sistema de recomendación y ejecutar posteriormente su entrenamiento, clasificación y, como su nombre indica, recomendación; "P1" para ejecutar las primeras pruebas, es decir generar los datos de tipo 1 y posteriormente hacer las pruebas pertinentes como se verá más adelante y "P2" para ejecutar las segundas pruebas. Las acciones aquí definidas se verán con más detalle en los próximos capítulos.

Ejecutar entrenamientos y prediccion...

Source on Save

```
# VARIABLES DE CONFIGURACION DE LA RED #

num_epocas <- 100 # veces que que se pasan a la red los datos de entrada para entrenarla
nr <- 0.4 # tasa de aprendizaje de la matriz de covarianza
alpha <- 0.7 # cantidad de varianza que describen las neuronas
num_neuronas <- 10 # numero de neuronas que componen la red

# INICIALIZACION DE LA RED #

E <- inicializarCentroides (num_neuronas, dim_datos)

R <- inicializarMatrizCovarianzas (num_neuronas, dim_datos)

BV <- calcularBV (R)

K <- array(dim_datos, num_neuronas) # dimension de las bases vectoriales de cada neurona

red <- list(centroides=E, covarianzas=R, bases_vectoriales=BV, dim_BV=K)

# ENTRENAMIENTO DE LA RED #

red <- entrenarRN(datos_normalizados, num_epocas, red, nr, alpha)

switch(accion,
       R0={
          clasificacion <- clasificarDatos(red, datos_normalizados)
          recomendacion <- matrix (0, nrow=dim_datos, ncol=num_total_datos)
          for(i in 1:num_total_datos){
             recomendacion[,i] <- recomendarProductos(datos, clasificacion, i)
          }
          write.table( recomendacion, paste(dir, "Recomendaciones.txt") )
       },
       P1={
          dir <- "Pruebas/Datos 1/Prueba 4/"
       },
       P2={
          clasificacion <- clasificarDatos(red, datos_normalizados)
          mostrarResultados2D(red,datos_normalizados, clasificacion)
          dir <- "Pruebas/Datos 2/Prueba 5/"
          write.table( resultado, paste(dir, "Resultados.txt") )
       },
       stop("No se ha encontrado la accion deseada")
)

# ALMACENAR VALORES DE LA RED #
write.table( red$centroides, paste(dir, "Centroides.txt") )
write.table( red$covarianzas, paste(dir, "Covarianzas.txt") )
write.table( red$bases_vectoriales, paste(dir, "Bases vectoriales.txt") )
write.table( red$dim_BV, paste(dir, "Dimension de las BV.txt") )

configuracion <- list ( epocas=num_epocas, neuronas=num_neuronas,
                        "tasa de aprendizaje"=nr, "cantidad de varianza"=alpha)
write.table( configuracion, paste(dir, "Configuracion.txt") )
```

Ilustración 3 Ejecutar entrenamientos y predicciones

En este último script, *"Ejecutar entrenamientos y predicciones"*, se lleva a cabo la inicialización de la configuración de la red y de las variables de la misma, así como el entrenamiento y según se hubiese indicado anteriormente se realizará la clasificación y recomendación del sistema o las pruebas de tipo 1 o 2. Una vez la red se ha entrenado y se ha realizado la acción que correspondiese, los resultados y la configuración y variables de la red se almacenan para tener constancia de cómo se han obtenido los resultados, para ello es necesario modificar la variable "dir" en cada ejecución de este script, para que así, se almacenen las variables y los resultados de cada prueba en el directorio correcto.

Estas tareas, llevadas a cabo en el sistema de recomendación, se pueden observar en su diagrama de flujo (ver ilustración 4).

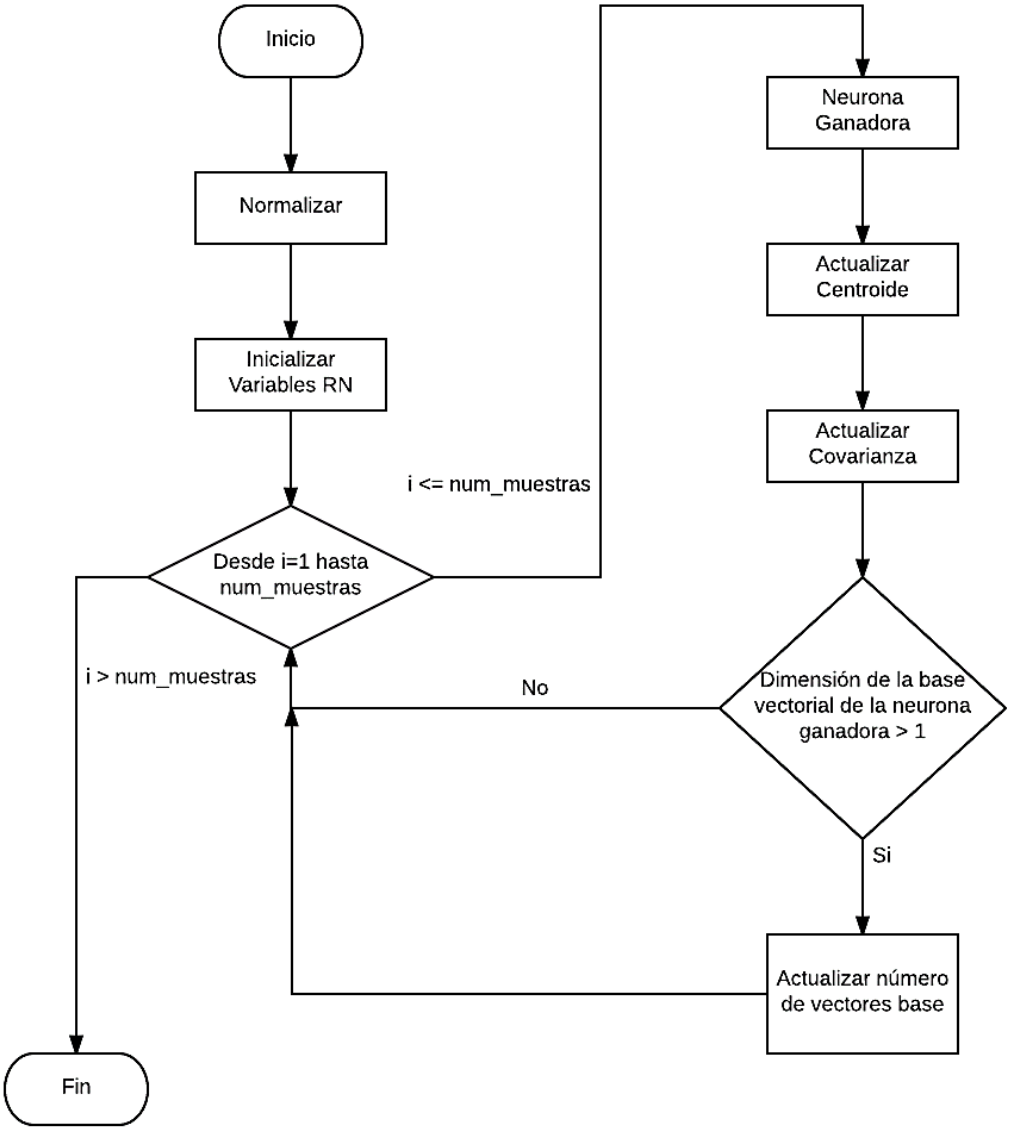


Ilustración 4. Diagrama de flujo del sistema de recomendación

Generación de los datos

Los datos finales del sistema de recomendación con los que se entrenará la red y que se clasificarán para dar la recomendación para cada usuario han sido cuidadosamente generados para que se asemejen a datos de compra reales. Esto se ha realizado con la función *"generarDatosRecomendacion"* (ver ilustración 5) que recibe como parámetros la dimensión de los datos y el tamaño del conjunto de datos (número de usuarios).

Generar datos.R

Source on Save

```r
# ------------------------------ GENERAR DATOS RECOMENDACION ---------------------------- #
#
# Funcion genera un conjunto de datos que simula las compras de un conjunto de usuarios
#
# Param dim: numero de columnas de la matriz de datos que equivale a los productos
# Param tam: numero de usuarios
#
# Return datos: datos que simulan las compras de usuarios
#
# ------------------------------------------------------------------------------------- #

generarDatosRecomendacion <- function (dim, tam){

  datos_aux <- matrix(0, nrow=dim, ncol=tam)

  for(i in 1:tam){

    datos_aux[,i] <-  sample(c(0,1), replace=TRUE, size=dim)

    # se inserta un dato aleatorio de una distribucion normal con media en 2
    datos_aux[datos_aux[,i]==1,i] <- rnorm(1,mean=2,sd=1)

  }

  datos_r <- ceiling(datos_aux) # valores enteros de los datos

  datos <- abs(datos_r) # valores absolutos de los datos

  return (datos)

}
```

Ilustración 5. Generar datos para recomendación

La función que genera los datos para el sistema de recomendación recibe la dimensión de la matriz que se debe generar, así como su tamaño y genera la matriz de datos inicialmente con ceros. Por cada columna, que correspondería a un vector de datos, se genera un vector aleatorio de ceros y unos y posteriormente se sustituyen los componentes de valor 1 por una muestra de una distribución normal de media 2 y desviación 1, esto se hace así puesto que normalmente los usuarios no suelen comprar o valorar todos los productos que se les ofrecen por lo que debe haber un mayor número de componentes nulas, esto se consigue con el primer vector de ceros y unos en el que al reemplazar sólo los unos, estos serán reemplazados por valores con media en 2 y desviación de 1 por lo que es probable que como resultado se obtengan más componentes nulas además de productos comprados o valorados 1, 2 o 3 veces. Se pueden obtener otros valores, pero estos tienen menor probabilidad por lo que aparecerán con menor frecuencia. La distribución uniforme devuelve datos reales por lo que, a la matriz de datos resultantes se le aplica la función *"ceiling"* para mantener sólo la componente entera de los valores de la matriz. Además,

se le aplica también la función *"abs"* para quedarnos sólo con valores positivos puesto que en compras no se pueden tener valores negativos y en el caso de las valoraciones no es recomendable dejar al usuario valorar negativamente.

Además de este conjunto de datos, se generan otros, de prueba, que se verán con detalle en el capítulo III.

Cabe destacar que, estos conjuntos de datos, forman una matriz en la que los usuarios se distribuyen por columnas y los productos por filas. Estos productos equivalen a la dimensión de los datos.

Preparación de los datos

Para que el sistema se entrene correctamente y posteriormente haga predicciones fiables, se debe realizar un preprocesamiento a los conjuntos de datos antes de pasarlos al sistema. El sistema PCACL, que se va a implementar, trabaja con vectores columna como datos de entrada. Por tanto, lo primero que se debe hacer es tener los datos correctamente almacenados en una matriz de vectores columnas. Así, los vectores de datos, corresponderán a una columna de la matriz.

La preparación de los datos puede variar dependiendo de cómo sean recogidos. En este caso los datos han sido generados en R siguiendo la estructura de una matriz de vectores columnas. Si los datos no estuviesen preparados y hubiesen sido recogido de un sistema real sería necesario encontrar una función que sea capaz de leer dichos datos y posteriormente almacenarlos en una matriz con las características definidas anteriormente.

Además, se tiene como objetivo introducir, al sistema, una simulación de datos reales y estos conjuntos de datos no suelen ser uniformes ya que los datos suelen ser de diferentes magnitudes. Es por ello que, los datos, van a ser normalizados puesto que, los valores del conjunto de datos que sean muy grandes influirán mucho en el resultado del sistema. Para normalizar el conjunto de datos se ha definido la función *"normalizar"* (ver ilustración 6) que, como su nombre indica, normaliza un conjunto de datos pasados como una matriz por parámetro.

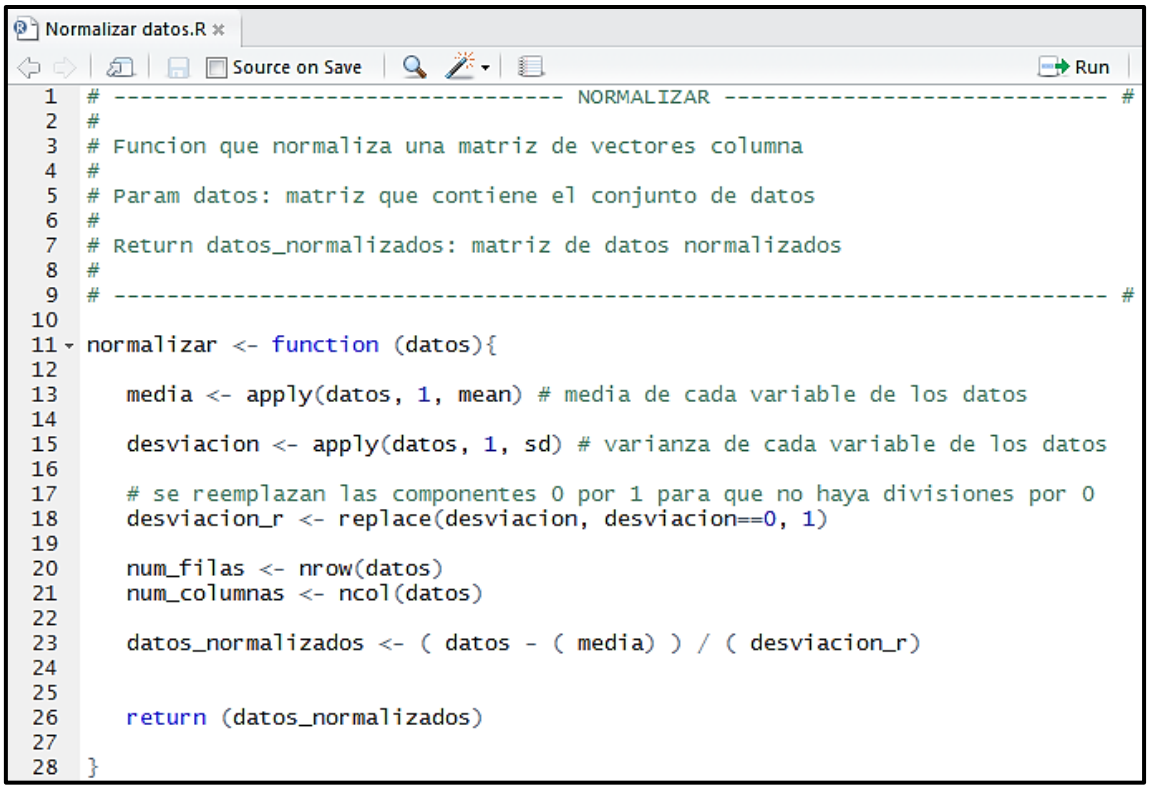


Ilustración 6. Normalizar datos

La tarea de normalizar tiene como finalidad que cada variable del conjunto de datos tenga una media de cero y una matriz de covarianzas de unos, para ello calcula la media o esperanza de los datos con la función *"apply"* de R, que es más eficiente que un bucle (Carmona, 2007). Del mismo modo se calcula la desviación típica de los datos. A esta función *"apply"* le pasamos la matriz de datos, indicamos la dimensión en la que queremos aplicar la función e indicamos que función se le quiere aplicar, para este caso utilizaremos la función *"mean"* para calcular la media de los datos y *"sd"* para calcular la desviación típica.

Una vez se han calculado la media y la varianza de cada dimensión normalizamos los datos siguiendo la ecuación $N = (X - (\boldsymbol{e} * I))/(\boldsymbol{v} * I)$ donde X es la matriz de datos, $\boldsymbol{e}$ es el vector de la media de cada dimensión, $\boldsymbol{v}$ es el vector de la desviación típica e I es la matriz identidad.

Esta matriz N de datos será la que se utilice como muestras de entrada para la red.

La red neuronal

Para comprender mejor el funcionamiento del entrenamiento de la red, primero, se va a describir cómo se obtiene la neurona ganadora de la red para un dato dado.

El cálculo de la neurona ganadora se realiza en el script *"Red neuronal"*. Este script contiene la función *"neuronaGanadora"* (ver ilustración 7) que recibe como

parámetros un vector de datos correspondiente al dato para el cuál se desea hallar la neurona ganadora y la red neuronal, es decir, una variable que contiene, en una lista, los datos de cada neurona de la red. Los datos que se almacenan de cada neurona en la lista de la red son: el centroide, la matriz de covarianza, los vectores base y el número de vectores base relevante de cada neurona.

Neurona ganadora.R

Source on Save

```r
# ------------------------------- NEURONA GANADORA ------------------------------ #
# Funcion que calcula la neurona ganadora de una red neuronal para un dato dado
#
# Param dato: dato de entrada
# Param red: lista con las variables de la red
## de las matrices de bases vectoriales de cada neurona
#
# Return NG: indice de la neurona ganadora
# ------------------------------------------------------------------------------ #
neuronaGanadora <- function(dato, red){

  NG <- NULL # neurona ganadora
  c <- NULL # variable auxiliar

  num_neuronas <- ncol(red$centroides)

   for(j in 1:num_neuronas){ # por cada neurona

      kj <- red$dim_BV[j] # dimension de la base vectorial de la neurona j

      base_vectorial <- red$bases_vectoriales[[j]] # base vectorial de la neurona j

      distancia <- dato - red$centroides[,j] # distancia entre el dato y la neurona

      z <- matrix(0,nrow=length(dato))

      # proyeccion ortonormal

      for (h in 1:kj){

         z <- z + ( t(distancia) %*% base_vectorial[,h] ) * base_vectorial[,h]
      }

      aux <- distancia - z

      res <- t(aux) %*% as.matrix(aux)

      c[j] <- res
   }

   NG <- which.min(c) # argumento minimo de c

   return (NG)
}
```

Ilustración 7. Neurona ganadora

Esta función celebra una competición entre las neuronas de la red y halla la neurona que mejor representa al dato de entrada que se le pasaba como parámetro.

La neurona ganadora será aquella que tenga el mínimo argumento para la proyección ortonormal de la distancia entre el dato y el centroide de la neurona en cuestión sobre la base vectorial $B = \{b_h | h = 1, \dots, k\}$, donde k equivale a la dimensión de los datos y B es la matriz ortonormal de los vectores base de la neurona j. Esta proyección ortonormal $\boldsymbol{z}_j^i(t) = Orth\left(\boldsymbol{x}^i(t) - \boldsymbol{e}_j(t), \boldsymbol{B}^j(t)\right)$, por

definición, se calcula como $z = \sum_{h=1}^{K}(\boldsymbol{v'} \cdot \boldsymbol{u_h}) * \boldsymbol{u_h}$, donde $\boldsymbol{v}$ el vector distancia, $\boldsymbol{u}$ la base ortonormal y K es la dimensión de la base (López Rubio, Ortiz de Lazcano Lobato, Muñoz Pérez, & Gómez Ruiz, 2004). Esto se basa en el PCA que se explicaba en el capítulo I.

Finalmente, la neurona ganadora será la que tenga el argumento mínimo que se obtiene con la función de R *"which.min()"*. (Carmona, 2007)

Entrenamiento de la red neuronal

Para realizar el entrenamiento de la red neuronal se han desarrollado diferentes scripts y funciones que se explicarán a continuación, así como su funcionamiento.

Como se podía observar en la ilustración 1, tras ejecutar el script *"Preparar entorno y datos"* (ver ilustración 2), que carga y preprocesa los datos, se definen, en el scrpit *"Ejecutar entrenamientos y predicciones"* (ver ilustración 3), las variables de la configuración de la red. Al definir aquí estas variables, resultará muy fácil variarlas durante las pruebas para encontrar los valores óptimos, es decir, con los que mejor aprende la red.

Las variables de la configuración de la red son: el número de épocas que se llevarán a cabo en el entrenamiento, la tasa de aprendizaje de la matriz de covarianza de las neuronas, la cantidad, α, de varianza que describen las neuronas y el número de neuronas que compondrán la red.

Es importante elegir un número de épocas adecuado puesto que cuanto mayor sea el valor, más se ajustará la red a los datos de entrenamiento y si la red se ajusta demasiado a los datos, podrá no hacer buenas predicciones para otros datos de entrada.

La tasa de aprendizaje de la matriz de covarianza influirá en cómo de rápido aprende una neurona, es decir, cuando una neurona es la ganadora para un dato de entrada, esta se actualiza para "acercarse" y representar mejor a ese dato de entrada. Si la tasa de aprendizaje es muy alta la neurona tenderá a representar muy bien al dato de entrada por lo que si era una equivocación la neurona intentará representar con mucha fuerza a un dato que no debería ser suyo y por lo tanto se cometerán más errores. En cambio, si la tasa de aprendizaje es muy baja, las neuronas aprenderán muy lentamente y podrían ser muy lejanas a datos que deberían representar.

No se ha mencionado la tasa de aprendizaje del centroide puesto que es una constante que sigue la ecuación $\eta_e = \frac{1}{1+M}$, donde M es el número de datos de entrada o usuarios en este caso, y se define antes de comenzar el entrenamiento como se verá más adelante.

El parámetro α (*"alpha"*) indica la cantidad de varianza mínima que deben describir las neuronas.

Elegir correctamente el número de neuronas de la red es muy importante puesto que determinará el número de grupos en los que se clasificarán los datos. Si este número es demasiado pequeño, la red agrupará datos que no tienen mucha correlación entre sí y a la hora de hacer las predicciones no será demasiado fiable. En cambio, si el número de neuronas es demasiado grande, se crearán grupos con pocos elementos o ninguno por lo que las predicciones que se hagan no serán relevantes.

Una vez se tienen los datos normalizados y las variables de la configuración de la red almacenadas, se inicializan las variables de cada neurona de la red siguiendo las indicaciones del algoritmo del PCACL (López Rubio, Ortiz de Lazcano Lobato, Muñoz Pérez, & Gómez Ruiz, 2004) y con la ayuda de las funciones que se verán a continuación y que están definidas en el script *"Inicializar Variables RN"*.

Primero se inicializan los centroides de las neuronas con la función *"inicializarCentroides"* que ha sido definida para realizar esta tarea (ver ilustración 8).

Inicializar variables de la RN.R

Source on Save

```
1  # ---------------------------- INICIALIZAR CENTROIDES ---------------------------- #
2  #
3  # Funcion que inicializa los centroides de cada neurona de la red
4  #
5  # Param num_neuronas: numero de centroides que se deben generar
6  # Param dim_datos: dimension de los datos que debe ser igual a la de los centroides
7  #
8  # Return E: matriz de vectores columnas que contiene los centroides
9  #
10 # ------------------------------------------------------------------------------- #
11 inicializarCentroides <- function ( num_neuronas, dim_datos ){
12
13    centroide <- matrix(0,dim_datos,num_neuronas)
14
15    for(j in 1:num_neuronas){  # por cada neurona
16
17       vector <- rnorm(dim_datos,0,0.3) # distribucion normal
18
19       centroide[,j] <- t(vector) # centroide inicial, e(0) de la neurona j
20    }
21
22    return (centroide)
23 }
```

Ilustración 8. Inicializar variables de la RN (centroides)

Esta función se encarga de generar una matriz de vectores columna. Estos vectores serán los vectores iniciales que representan el centroide, $\boldsymbol{e}_j(0)$, de cada neurona j. Los centroides iniciales debe estar compuestos por pequeños valores aleatorios tanto positivos como negativos para que, al inicio, las neuronas se encuentren cercanas a los datos de entrada normalizados.

A continuación, se inicializan las matrices de covarianza de cada neurona con la ayuda de la función *"inicializarMatrizCovarianzas"* que se ha definido para generar y almacenar en una lista las matrices de covarianza de cada neurona de la red. Esta función genera, por cada neurona j, la matriz de covarianza inicial $\boldsymbol{R}_j(0)$, de forma que resulte una matriz no negativa, simétrica y aleatoria con los elementos de la diagonal principal cercanos a uno y el resto cercanos a cero (ver ilustración 9).

Inicializar variables de la RN.R

Source on Save | Run

```
# ------------------------------- INICIALIZAR MATRIZ DE COVARIANZAS ------------------------------- #
#
# Funcion que inicializa las matrices de covarianzas de cada neurona de la red
#
# Param num_neuronas: numero de matrices que se deben generar
# Param dim_datos: dimension de los datos que coincide con la dimension de la matriz
#
# Return R: lista de las matrices de las neuronas
#
# ------------------------------------------------------------------------------------------------- #
inicializarMatrizCovarianzas <- function ( num_neuronas, dim_datos ){

   covarianzas <- list()

   for(j in 1:num_neuronas){  # por cada neurona

      num_componentes <- dim_datos*dim_datos # numero de componentes de la matriz

      val_diagonal <- rnorm(dim_datos,1,0.1) # valores de la diagonal de la matriz

      # numero de elementos que hay que generar sin contar la diagonal
      num_elementos <- ( (dim_datos*dim_datos) - dim_datos ) / 2

      valores <- rnorm(num_elementos,0,0.1) # valores del resto de la matriz

      # valores absolutos #
      componentes <- abs(valores)
      diagonal <- abs(val_diagonal)

      covarianza <- matrix(0,dim_datos,dim_datos) # generacion de la matriz

      covarianza[lower.tri(covarianza)] <- componentes #componentes inferiores

      covarianza[upper.tri(covarianza)] <- t(covarianza)[upper.tri(covarianza)] #componentes superiores

      diag(covarianza) <- array(diagonal,dim_datos) # componentes de la diagonal

      covarianzas[j] <-list(covarianza)
   }
   return (covarianzas)
}
```

Ilustración 9. Inicializar variables de la RN (matrices de covarianzas)

Para generar dicha matriz se obtienen sus componentes de la diagonal principal con una distribución normal aleatoria de media 1 y desviación 0.1 y el resto de elementos seguirán una distribución también normal con desviación 0.1 pero con media 0. Como dicha matriz debe ser una matriz no negativa, se hayan los valores absolutos de los valores obtenidos. Para que sea simétrica se utilizan las funciones de R *"lower"* y *"upper"* con las que primero se introducen los elementos en la parte inferior de la matriz y posteriormente se copia la parte inferior de la matriz a la parte superior. También se introducen los valores de la diagonal principal obtenidos de la distribución normal de media uno. Así, resulta la matriz de covarianza simétrica aleatoria no negativa que se deseaba.

A continuación, se calcula la matriz de bases vectoriales de cada neurona con la función *"calcularBV"*, definida en el script *"Calcular bases vectoriales"* (ver ilustración 10). Estas bases vectoriales son bases vectoriales ortonormales

compuestas por los autovectores de la matriz de covarianza de cada neurona. Estas matrices de bases vectoriales tendrán un papel muy importante a la hora de calcular la neurona ganadora como se ha visto al principio del capítulo en el apartado “La red neuronal”.

Calcular bases vectoriales.R

Source on Save | Run

```
# ---------------------------- CALCULAR BASES VECTORIALES ------------------------------- #
#
# Funcion que calcula las bases vectoriales de las matrices de covarianzas de cada neurona
#
# Param covarianzas: lista de las matrices de covarianzas de la red
#
# Return BV: matriz de vectores columnas que contiene los vectores base
#
# ------------------------------------------------------------------------------------- #
calcularBV <- function (covarianzas) {

   num_neuronas <- length(covarianzas) # numero de neuronas de la red

   BV <- list() # variable que almacena la lista de las bases vectoriales de cada neurona

   for(s in 1:num_neuronas){  # por cada neurona

      # --------- Base Vectorial Ortonormal de la neurona s --------- #

      e <- eigen(covarianzas[[s]])

      BV[s] <- list(e$vectors)
   }

   return (BV)
}
```

Ilustración 10. Calcular bases vectoriales

La matriz de vectores base de cada neurona, como se puede observar en la ilustración anterior, se halla con la función de R *“eigen()”*, la cual calcula los autovectores y autovalores de la matriz dada (R Development Core Team, 2000). Cabe destacar que esta función devuelve la matriz de vectores base ordenados de mayor a menor autovalor.

En este momento sólo interesa almacenar los autovectores puesto que, como se ha mencionado antes, serán necesarios en el cálculo de la neurona ganadora.

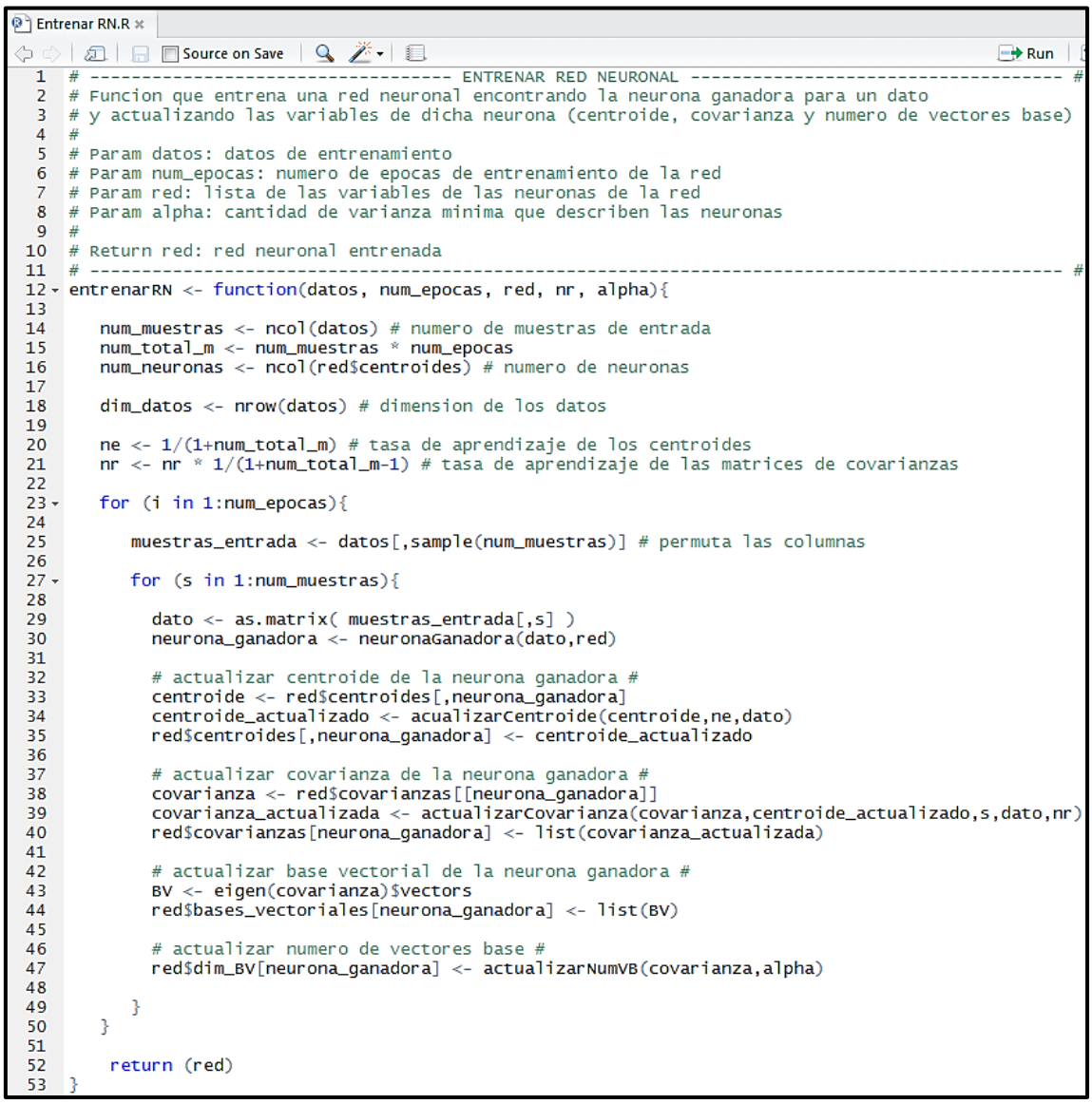
Estas variables anteriormente definidas se almacenan en una lista denominada *“red”* (ver ilustración 3), como se ha descrito anteriormente. Esta variable se define para tener una mayor facilidad de acceso a las variables de la red neuronal.

Tras la inicialización, de las neuronas de la red, y el almacenamiendo de la configuración y las variables necesarias, se procede a entrenar la misma. La tarea de entrenar la red neuronal la realiza la función *“entrenarRN”* (ver ilustración 11).

Esta función recibe como parámetros los datos de entrenamiento, el número de épocas[2], la red neuronal con los datos iniciales que se han obtenido en el paso

[2] Una época es el proceso que se lleva a cabo al entrenar la red neuronal con todas las muestras de entrada

anteriormente descrito y las constantes η_R que es la tasa de aprendizaje de las matrices de covarianza y α (alpha) que es la cantidad de varianza mínima que describirán las neuronas.




```r
# ---------------------------------- ENTRENAR RED NEURONAL ---------------------------------- #
# Funcion que entrena una red neuronal encontrando la neurona ganadora para un dato
# y actualizando las variables de dicha neurona (centroide, covarianza y numero de vectores base)
#
# Param datos: datos de entrenamiento
# Param num_epocas: numero de epocas de entrenamiento de la red
# Param red: lista de las variables de las neuronas de la red
# Param alpha: cantidad de varianza minima que describen las neuronas
#
# Return red: red neuronal entrenada
# ------------------------------------------------------------------------------------------- #
entrenarRN <- function(datos, num_epocas, red, nr, alpha){

   num_muestras <- ncol(datos) # numero de muestras de entrada
   num_total_m <- num_muestras * num_epocas
   num_neuronas <- ncol(red$centroides) # numero de neuronas

   dim_datos <- nrow(datos) # dimension de los datos

   ne <- 1/(1+num_total_m) # tasa de aprendizaje de los centroides
   nr <- nr * 1/(1+num_total_m-1) # tasa de aprendizaje de las matrices de covarianzas

   for (i in 1:num_epocas){

      muestras_entrada <- datos[,sample(num_muestras)] # permuta las columnas

      for (s in 1:num_muestras){

        dato <- as.matrix( muestras_entrada[,s] )
        neurona_ganadora <- neuronaGanadora(dato,red)

        # actualizar centroide de la neurona ganadora #
        centroide <- red$centroides[,neurona_ganadora]
        centroide_actualizado <- acualizarCentroide(centroide,ne,dato)
        red$centroides[,neurona_ganadora] <- centroide_actualizado

        # actualizar covarianza de la neurona ganadora #
        covarianza <- red$covarianzas[[neurona_ganadora]]
        covarianza_actualizada <- actualizarCovarianza(covarianza,centroide_actualizado,s,dato,nr)
        red$covarianzas[neurona_ganadora] <- list(covarianza_actualizada)

        # actualizar base vectorial de la neurona ganadora #
        BV <- eigen(covarianza)$vectors
        red$bases_vectoriales[neurona_ganadora] <- list(BV)

        # actualizar numero de vectores base #
        red$dim_BV[neurona_ganadora] <- actualizarNumVB(covarianza,alpha)

      }
   }

    return (red)
}
```

Ilustración 11. Entrenar red neuronal

Esta función lleva a cabo lo siguiente:

Por cada época, se permutan los datos de entrada, es decir, se intercambian las columnas de la matriz para que, en cada época, los datos entren en la red en un orden distinto.

Una vez permutadas las columnas de la matriz de datos, se realiza un entrenamiento en línea, es decir, uno a uno se le pasan los datos a la red, se halla la neurona ganadora y se actualizan los parámetros de dicha neurona. Estos pasos se realizan con la ayuda de la función *"neuronaGanadora"*

(*explicada en el capítulo III*) y las funciones *"actualizarCentroide", "actualizarCovarianza"* que se explican a continuación.

Una vez se sabe qué neurona es la ganadora, se actualizan sus parámetros. Primero se actualiza el centroide (ver ilustración 12) siguiendo la ecuación $\boldsymbol{e}_c(t+1) = \boldsymbol{e}_c(t) + \eta_e\big(\boldsymbol{x} - \boldsymbol{e}_c(t)\big)$ (López Rubio, Ortiz de Lazcano Lobato, Muñoz Pérez, & Gómez Ruiz, 2004), donde $\boldsymbol{e}_c$ es el centroide de la neurona ganadora, $\eta_e = \frac{1}{1+m}$ la tasa de aprendizaje del centroide, m el número de muestras total que se le pasarán a la red para su entrenamiento y $\boldsymbol{x}$ del dato de entrada.

Actualizar variables de la RN.R

Source on Save

```r
# ---------------------- ACTUALIZAR CENTROIDE ---------------------- #
#
# Funcion que actualiza el centroide de una neurona
#
# Param centroide: centroide actual de la neurona
# Param dato: dato de entrada
#
# Return centroide_actualizado: centroide de la neuorna actualizado
#
# ------------------------------------------------------------------ #

acualizarCentroide <- function(centroide,ne,dato){

   distancia <- dato - centroide

   centroide_actualizado <- as.matrix(centroide + ne*(distancia))

   return (centroide_actualizado)
}
```

Ilustración 12. Actualizar variables de la RN (centroide)

A continuación, se actualiza la matriz de covarianza de la misma neurona (ver ilustración 13) que se calcula como $\boldsymbol{R}_c(t+1) = \boldsymbol{R}_c(t) + \eta_R[\boldsymbol{A}(t+1) - \boldsymbol{R}_c(t)]$ donde $\boldsymbol{R}_c$ es la matriz de covarianza de la neurona ganadora, η_R la tasa de aprendizaje y $\boldsymbol{A}(t+1) = (\boldsymbol{x} - \boldsymbol{e_c}(t+1))(\boldsymbol{x} - \boldsymbol{e_c}(t+1))^t$. (López Rubio, Ortiz de Lazcano Lobato, Muñoz Pérez, & Gómez Ruiz, 2004)

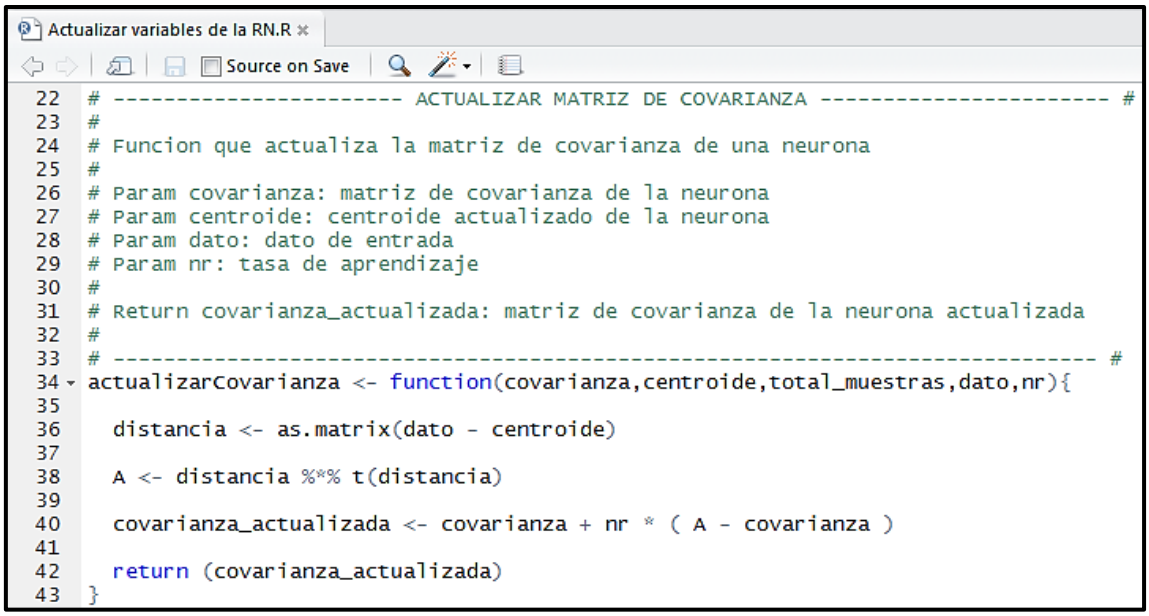


Ilustración 13. Actualizar variables de la RN (matriz de covarianza)

Por último, se actualiza la base vectorial de la neurona ganadora (ver ilustración 11, líneas 60-62) y se actualiza el número de vectores base de la misma con la ayuda de la función *"actualizarNumVB"* (ver ilustración 14).

En el modelo PCACL, el número de vectores base de la matriz de covarianza de una neurona se obtiene de la ecuación:

$K_j = min\{Z\epsilon\{0,1,\dots,D\} \mid \sum_p^Z \lambda_j^p\,(t) \geq \alpha\, trace(\boldsymbol{R}_j(t))\}$, donde j es el índice de la neurona, D es la dimensión máxima de la base vectorial, λ_j^p es el autovalor p de la matriz de covarianza de la neurona j y α es la cantidad de varianza mínima que se desea que describan las neuronas (López Rubio, Ortiz de Lazcano Lobato, Muñoz Pérez, & Gómez Ruiz, 2004) y ha sido indicada al comienzo del sistema en el script *"Ejecutar entrenamiento y recomendación"* (ver ilustración 3).

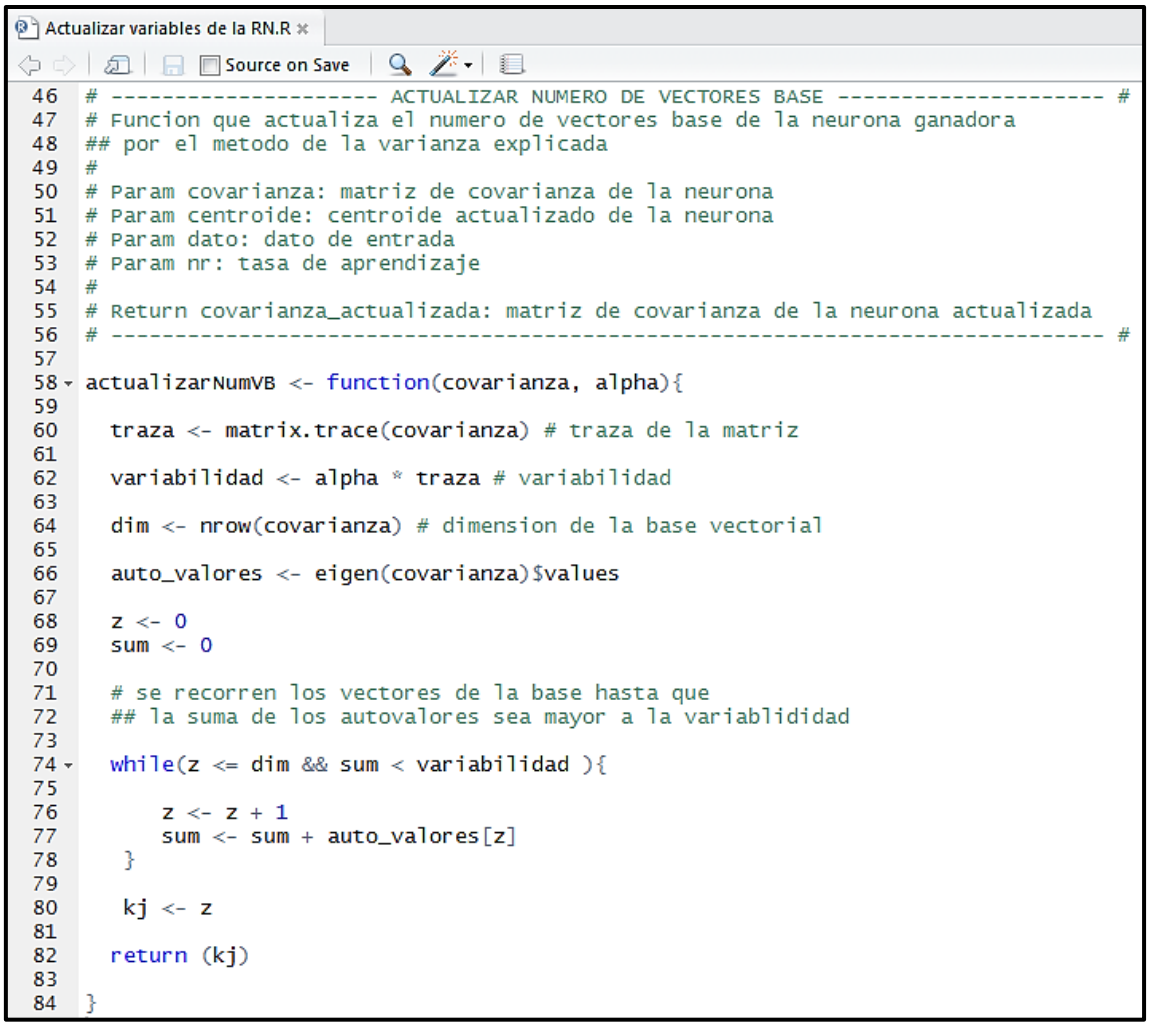


Ilustración 14. Actualizar variables de la RN (número de vectores base)

Capítulo III. Pruebas de la red neuronal

Para probar el correcto funcionamiento del algoritmo implementado, se han generado en R dos tipos de datos artificiales. Estos datos servirán para comprobar que el sistema funciona de la forma que se esperaba y para determinar si es capaz de detectar la dimensión intrínseca de los datos, es decir, se verificará si la clasificación, tras el entrenamiento, es correcta.

Las pruebas se realizan desde el script *"Principal"* y se verán con más detalle cada una en los próximos apartados.

Se ha definido una función denominada *"clasificarDatos"* (ver ilustración 15) que, dada una red y una matriz de datos, devuelve un array con la neurona ganadora para cada vector dato según su índice.

Clasificar datos.R

Source on Save

```r
# ------------------------------ CLASIFICAR DATOS ------------------------------ #
#
# Funcion que clasifica un conjunto de datos
#
# Param red: red neuronal entrenada
# Param datos: conjunto de datos para clasificar
#
# Return NG: array que contiene en cada indice la neurona representante del dato
#
# ------------------------------------------------------------------------------ #

clasificarDatos <- function (red, datos) {

  num_neuronas <- ncol(red$centroides) # numero de neuronas de la red

  dim_datos <- nrow(datos) # dimension de los datos

  total_datos <- ncol(datos) # numero total de datos


  # Se halla la neurona ganadora para cada dato #

  NG <- array(0,total_datos)

  for(i in 1:total_datos){

    dato <- as.matrix( datos[,i] )

    NG[i] <- neuronaGanadora(dato,red)

  }

  return (NG)

}
```

Ilustración 15. Clasificar datos

Los datos obtenidos de pruebas se encuentran en la carpeta *"Pruebas"* dentro del proyecto "Sistema de recomendación" puesto que es el ámbito en el que se trabaja en este proyecto de R. En esta carpeta pruebas se diferencias las mismas por los datos que se han utilizado para dichas pruebas y en las que se encontrará una hoja con los datos que generó la red, además de la gráfica del conjunto de datos generado y las carpetas referentes a las pruebas realizadas al conjunto de datos que se trata. En estas carpetas de prueba se puede encontrar la configuración de las variables de la red, las variables de cada neurona separadas en centroides, matriz de covarianza, bases vectoriales y número de vectores base de cada neurona. Además de, los resultados de la clasificación si procede.

A continuación, se muestra cada tipo de prueba y cómo se han generado los datos para la misma además de sus resultados y el análisis de los mismos. Tras el entrenamiento de la red se estudiarán sus resultados y predicciones en cada caso para comprobar si la red halla la dimensión intrínseca de los datos y si realiza una buena clasificación de los mismos.

Como se verá más adelante, el conjunto de datos 2 tiene como objetivo que las neuronas se adapten a dos grupos de datos y los clasifiquen correctamente. Para poder visualizar está clasificación y comprobar más rápidamente si es o no correcta, se ha definido la función *"mostrarResultados2D"* (ver ilustración 16).

Mostrar resultados.R

Source on Save

```r
# ------------------------------ MOSTRAR RESULTADOS 2D ------------------------------ #
#
# Funcion que muestra una grafica con las neuronas de una red (centroide y covarianza)
# y un conjunto de datos clasificados por colores
#
# Param red: red neuronal
# Param datos: datos que se han clasificados
# Param resultados: array con los resultados de la clasificacion de los datos
#
# ----------------------------------------------------------------------------------- #
mostrarResultados2D <- function(red,datos,resultados){

   num_neuronas <- ncol(red$centroides) # numero de neuronas de la red
   total_datos <- ncol(datos) # total de datos para clasificar

   # se obtienen los datos de cada eje como un vector
   datosX <- datos[1,]
   datosY <- datos[2,]

   # conjunto de puntos que se van a graficar
   X <- c ( red$centroides[1,], datosX)
   Y <- c ( red$centroides[2,], datosY)

   # rango de los ejes del grafico
   cons <- 1
   max_x <- max(X) + cons
   max_y <- max(Y) + cons
   min_x <- min(X) - cons
   min_y <- min(Y) - cons

   # inicializacion del grafico
   win.graph()

   plot(x=NULL, y=NULL, xlab="P1", ylab="P2", xlim=c(min_x,max_x), ylim=c(min_y,max_y),
        main="Usuarios clasificados")

   #inicializar valores de la leyenda
   leyenda <- c()
   colores <- c()

   for(j in 1:num_neuronas){ # por cada neurona

      centroide <- red$centroides[,j] # centroide de la neurona j

      covarianza <- red$covarianzas[[j]] # covarianza de la neurona j

      RR <- chol(covarianza)  # descomposicion de Cholesky

      angulos <- seq(0, 2*pi, length.out=200) # angulos de la elipse

      elipse <- cbind(cos(angulos), sin(angulos)) %*% RR

      elipseCent <- sweep(elipse, 2, centroide, "+")

      color <- j+1

      # se grafica la elipse
      points(elipseCent, type="l", lwd=2, asp=1,col=color)

      # se grafica el centroide
      points(centroide[1], centroide[2], pch=4, lwd=2, col=color)

      # se grafica el nombre del grupo
      desp <- 0.2
      posX <- centroide[1]+desp
      posY <- centroide[2]+desp

      # se genera la leyenda
      leyenda <- c(leyenda,paste("Neurona ",j))
      colores <- c(colores,color)
   }

   for(i in 1:total_datos){ # por cada dato

      # color del grupo donde esta clasificado el dato i
      color <- colores[resultados[i]]

      # se grafica el dato i
      points(datos[1,i],datos[2,i], cex=1.5, pch=21, bg=color, col=color)

   }

   # se inserta la leyenda
   legend("topright", legend=leyenda, col=colores, pch=15, title="Grupos")
}
```

Ilustración 16. Mostrar resultados 2D

Esta función muestra en una gráfica las neuronas en dos colores distintos marcado con una “x” el centroide y con una elipse la matriz de covarianza. Así mismo, los datos se representan con puntos del color de la neurona que los ha clasificado.

Conjunto de datos 1

Para esta primera prueba se ha generado un conjunto de datos de 10 dimensiones en las que las dimensiones 4 y 5 corresponden a un conjunto de datos que forman un arco en dos dimensiones con radio 10 y el resto de componentes son 0. Con esta prueba se pretende comprobar que la red haya correctamente la dimensión intrínseca de los datos, es decir, debería obtenerse un número de vectores base igual a 2 puesto que sólo hay dos dimensiones que son relevantes.

Para generar estos datos se ha definido la función *“generarDatosArco”* (ver ilustración 17) que dado un radio y un límite genera los datos de un arco con el radio dado y en el rango $\mp lim$, donde lim es el límite dado. Posteriormente se introducen estos datos en las dimensiones 4 y 5 de un conjunto de datos de n dimensiones, donde n viene dado por la variable *“dim”* en el que el resto de componentes tienen valor 0.

Generar datos.R

Source on Save

```
# ------------------------------- GENERAR DATOS ARCO ------------------------------ #
#
# Funcion genera un conjunto de datos que forman un arco en dos de sus dimensiones
## y dibuja los datos en una grafica
#
# Param radio: radio del arco
# Param lim: limites del arco
# Param dim: dimension de los datos
#
# Return datos: conjunto de datos con los datos del arco en las dimensiones 4 y 5
#
# ------------------------------------------------------------------------------- #

generarDatosArco <- function(radio,lim,dim){

  abs_lim <- abs(lim) # valor absoluto

  inf <- -abs_lim #limite inferior

  sup <- abs_lim # limite superior

  paso <- abs_lim/100 # diferencia entre cada dato

  # conjunto de datos 2D

  x <- seq(from=inf, to=sup, by=paso)

  y <- sqrt(radio*radio - x*x);

  tam_datos <- length(x)


  datos <- matrix(0,dim,tam_datos) # conjunto de datos de dimension 10

  datos[4,] <- x
  datos[5,] <- y

  # GRAFICO DE DATOS
  win.graph()

  plot(x=x, y=y, xlab="P4", ylab="P5", main="Usuarios de prueba 1", col="blue")

  return(datos)

}
```

Ilustración 17. Generar datos (arco)

El conjunto generado para esta prueba se encuentra guardado en la hoja *"Datos 1"* dentro del fichero de pruebas con el mismo nombre. La representación gráfica del arco contenido en las dimensiones 4 y 5 de estos datos se puede observar en la siguiente ilustración (ver ilustración 18).

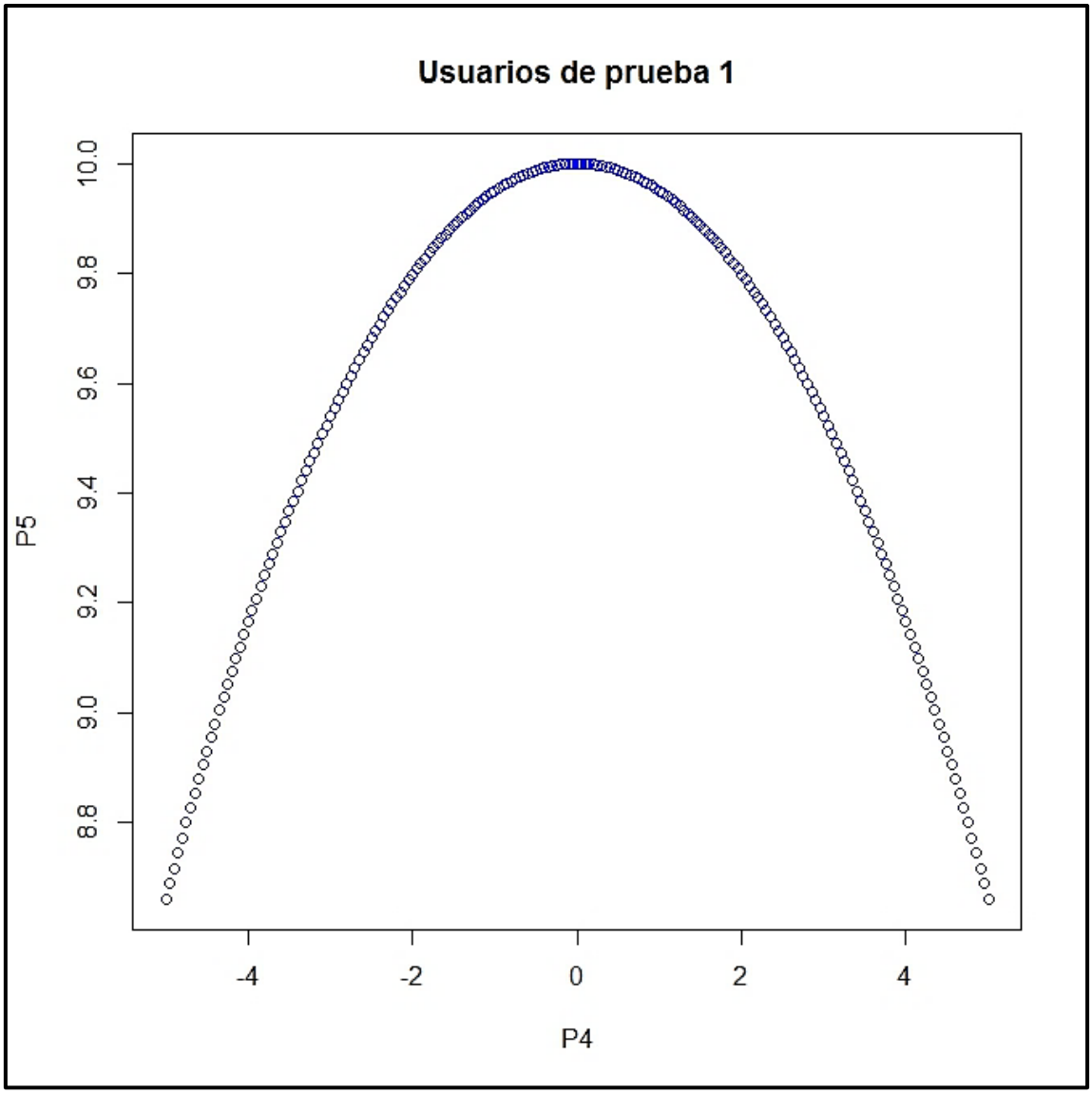


Ilustración 18. Representación gráfica del arco

A continuación, se muestran las pruebas realizadas a este conjunto de datos.

Prueba 1

La configuración utilizada para esta prueba es la siguiente:

- Número de épocas: 100
- Tasa de aprendizaje de la matriz de covarianza: 0.6
- Cantidad de varianza que describen las neuronas: 0.7
- Número de neuronas: 2

En este caso lo más importante será la cantidad de varianza que describen las neuronas, se debe recordar que el conjunto de datos era un conjunto de dimensión 10 y el análisis de componentes principales que se le realiza a cada neurona debería dar un resultado de 2 como número de vectores base puesto que sólo son dos las dimensiones relevantes.

Las dos neuronas de la red han dado un resultado de 6 como número de vectores base, esto nos indica que, en una segunda prueba, se debe ajustar mejor la cantidad de varianza que describen las neuronas.

Prueba 2

La configuración de esta prueba es la siguiente:

- Número de épocas: 100
- Tasa de aprendizaje de la matriz de covarianza: 0.6
- Cantidad de varianza que describen las neuronas: 0.3
- Número de neuronas: 2

En esta segunda prueba se ha disminuido la cantidad de varianza a 0.3 y se ha obtenido el resultado deseado de 2 vectores base en cada neurona.

De estos resultados se puede concluir que la red halla correctamente la dimensión intrínseca de los datos.

Conjunto de datos 2

Este conjunto de datos se ha generado de forma aleatoria con la ayuda de la función definida como *“generarDatosSeparados”* (ver ilustración 19).

Generar datos.R

Source on Save | Run

```r
# ------------------------------- GENERAR DATOS SEPARADOS ------------------------------- #
#
# Funcion genera un conjunto de datos en 2D que esta formado por dos conjuntos diferenciados
## y dibuja los datos en una gr?fica y los datos normalizados en otra
#
# Param rango: rango de los datos
# Param multiplo: valor por el que se multiplican los datos en la componente y
#
# Return datos: conjunto de datos generados
# ---------------------------------------------------------------------------------------- #
generarDatosSeparados <- function(rango,multiplo){

   min_1 <- rango * (-2)
   max_1 <- rango * (-1)
   min_2 <- rango
   max_2 <- rango *2

   x1 <- seq(from=min_1, to=max_1, by=1)
   x2 <- seq(from=min_2, to=max_2, by=1)
   y1 <- runif(n=rango+1, min=min_1, max=max_1)*multiplo
   y2 <- runif(n=rango+1, min=min_2, max=max_2)*multiplo

   tam_datos <- length(x1)

   conjunto_1 <- matrix(0,nrow=2,ncol=tam_datos)
   conjunto_1[1,] <- x1
   conjunto_1[2,] <- y1

   conjunto_2 <- matrix(0,nrow=2,ncol=tam_datos)
   conjunto_2[1,] <- x2
   conjunto_2[2,] <-y2

   datos <- cbind(conjunto_1,conjunto_2)
   datos_normalizados <- normalizar(datos)

   # GRAFICO DE DATOS #

   win.graph()

   plot(x=datos[1,], y=datos[2,], xlab="P1", ylab="P2",
        xlim=c(min_1,max_2), ylim=c(min_1*multiplo,max_2*multiplo),
        main="Usuarios de prueba 2", col="red")

   # GRAFICO DE DATOS NORMALIZADOS #

   win.graph()

   plot(x=datos_normalizados[1,], y=datos_normalizados[2,],
        xlim=c(-1.5,1.5),ylim=c(-1.5,1.5), xlab="P1", ylab="P2",
        main="Usuarios de prueba 2 normalizados", col="red",pch=3)

   return(datos)
}
```

Ilustración 19. Generar datos separados

Esta función recibe un rango y un múltiplo, el rango es el valor que define dónde se encontrarán los conjuntos de datos en el eje X y en el eje Y se multiplican por la variable *"multiplo"*. Los datos del eje X se generan de forma secuencial con saltos de uno en uno, el primer grupo estará contenido entre el valor negativo del doble del rango y el valor negativo del rango y el segundo grupo entre el valor del rango y el doble del mismo. En cambio, para generar los datos del eje Y se ha utilizado la función de R *"runif"* que genera datos aleatorios siguiendo una distribución uniforme de n datos dentro de un rango dado, los resultados de esta distribución se multiplican por la variable *"multiplo"* para realizar una mayor separación de ambos conjuntos de datos. Estos datos quedarán bien diferenciados dentro del primer y tercer cuadrante (ver ilustración 20).

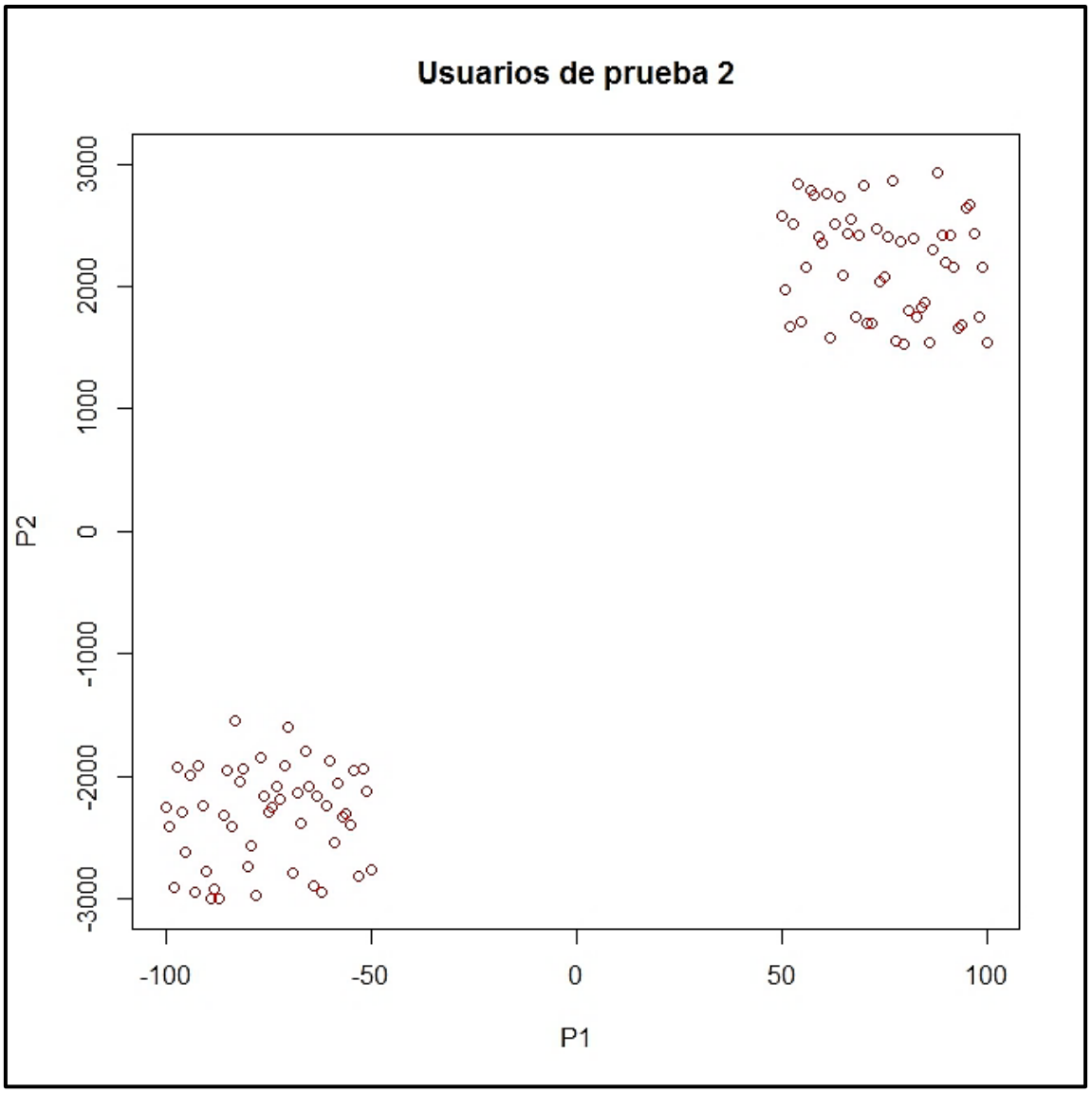


Ilustración 20. Gráfica del conjunto de datos separados

La función anteriormente definida (ver ilustración 19), además de generar la gráfica para el conjunto de datos generado, los normaliza y genera otra gráfica para representar el conjunto de datos normalizado (ver ilustración 21)

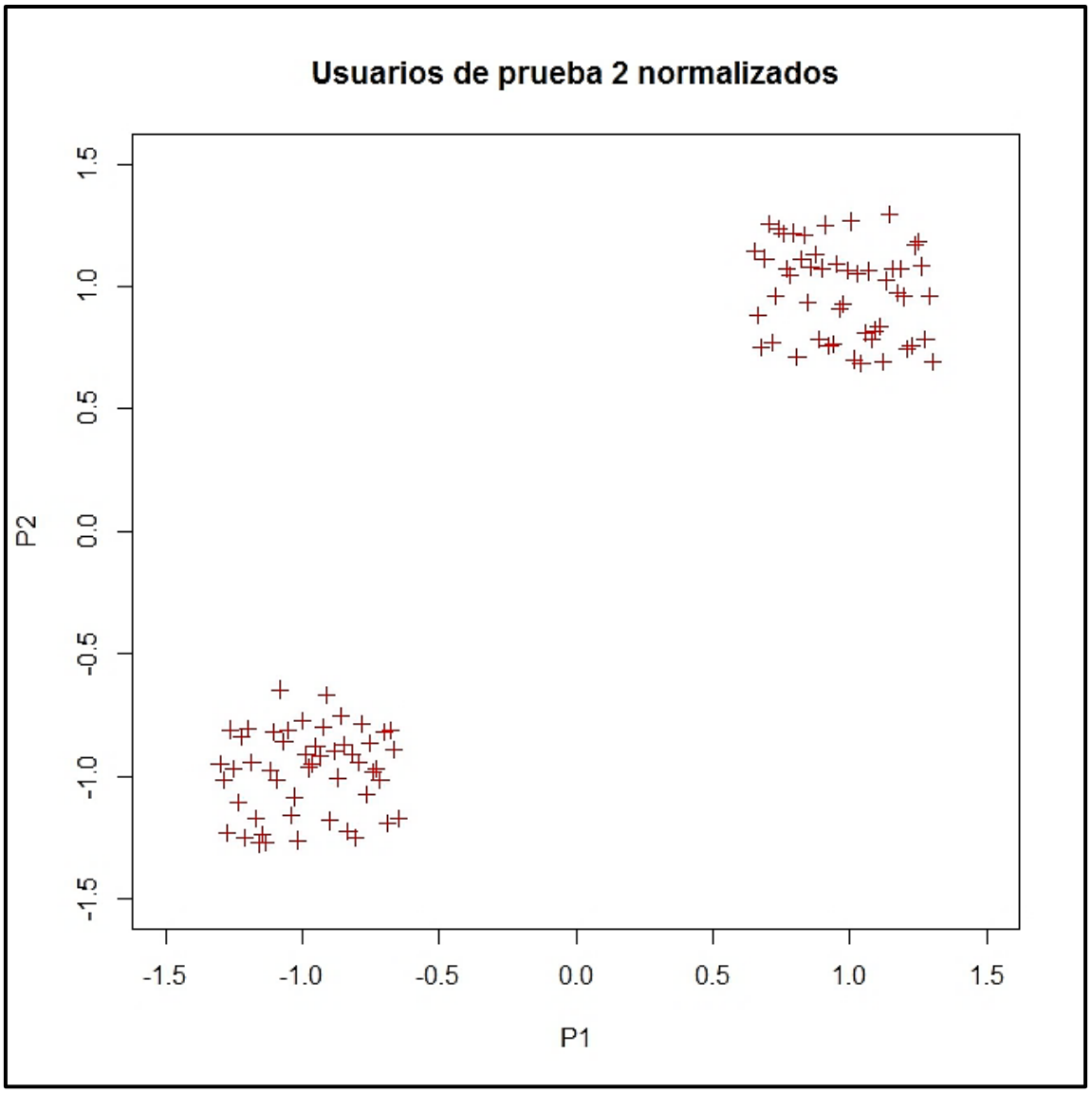


Ilustración 21. Gráfica del conjunto de datos separados normalizado

A continuación, se muestran las pruebas realizadas para este conjunto de datos.

Prueba 1

La configuración de esta primera prueba es la siguiente:

- Número de épocas: 100
- Tasa de aprendizaje de la matriz de covarianza: 0.6
- Cantidad de varianza que describen las neuronas: 0.9
- Número de neuronas: 2

Como se podía observar en las imágenes anteriores, los datos se encuentran en dos grupos claramente diferenciados por lo que se ha escogido 2 como el número total de neuronas de la red y como se desea que las neuronas mantengan todas las dimensiones se ha escogido 0.9 como la cantidad de varianza que se desea que describan las neuronas.

La clasificación de los datos normalizados tras en el entrenamiento, se puede observar en la siguiente ilustración (ver ilustración 22).

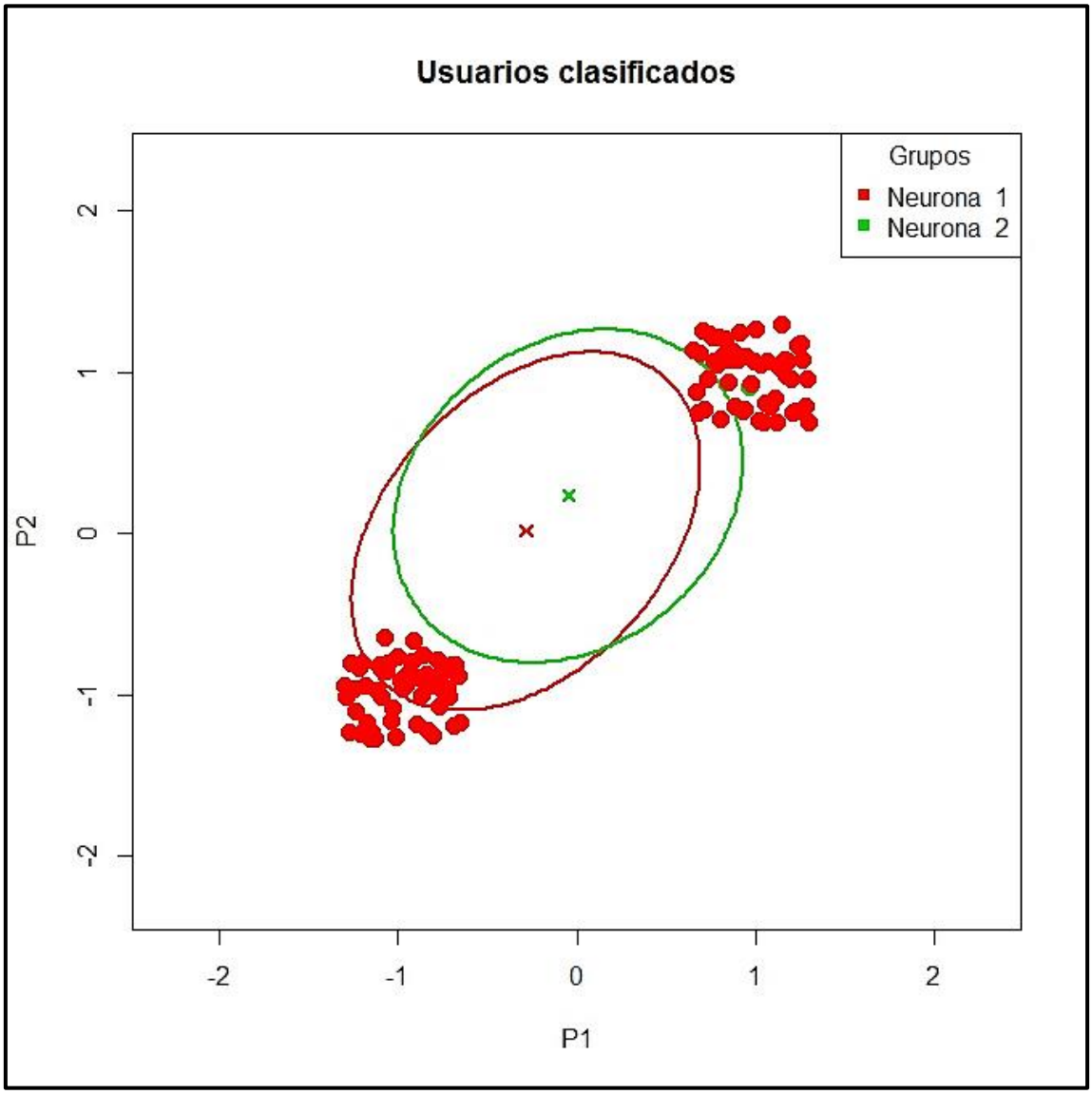


Ilustración 22. Gráfica de clasificación (datos 2, prueba 1)

De la ilustración anterior se puede concluir que las neuronas no han aprendido bien, puesto que se deseaba que cada neurona representase un conjunto de datos diferente, pero al estar ambas inicializadas con una media cercana a 0 y al tener los datos una media de 0 (por estar normalizados), ambas neuronas, no han podido aprender correctamente en este entrenamiento, sólo una se ha llevado, prácticamente, el aprendizaje de todos los datos del conjunto.

En las próximas pruebas se intentará mejorar la clasificación de estos datos mejorando la configuración de la red.

Prueba 2

La configuración de esta prueba es la siguiente:

- Número de épocas: 100
- Tasa de aprendizaje de la matriz de covarianza: 0.1
- Cantidad de varianza que describen las neuronas: 0.1
- Número de neuronas: 2

El resultado de clasificar los datos tras el entrenamiento se puede observar en la siguiente ilustración (ver ilustración 23).

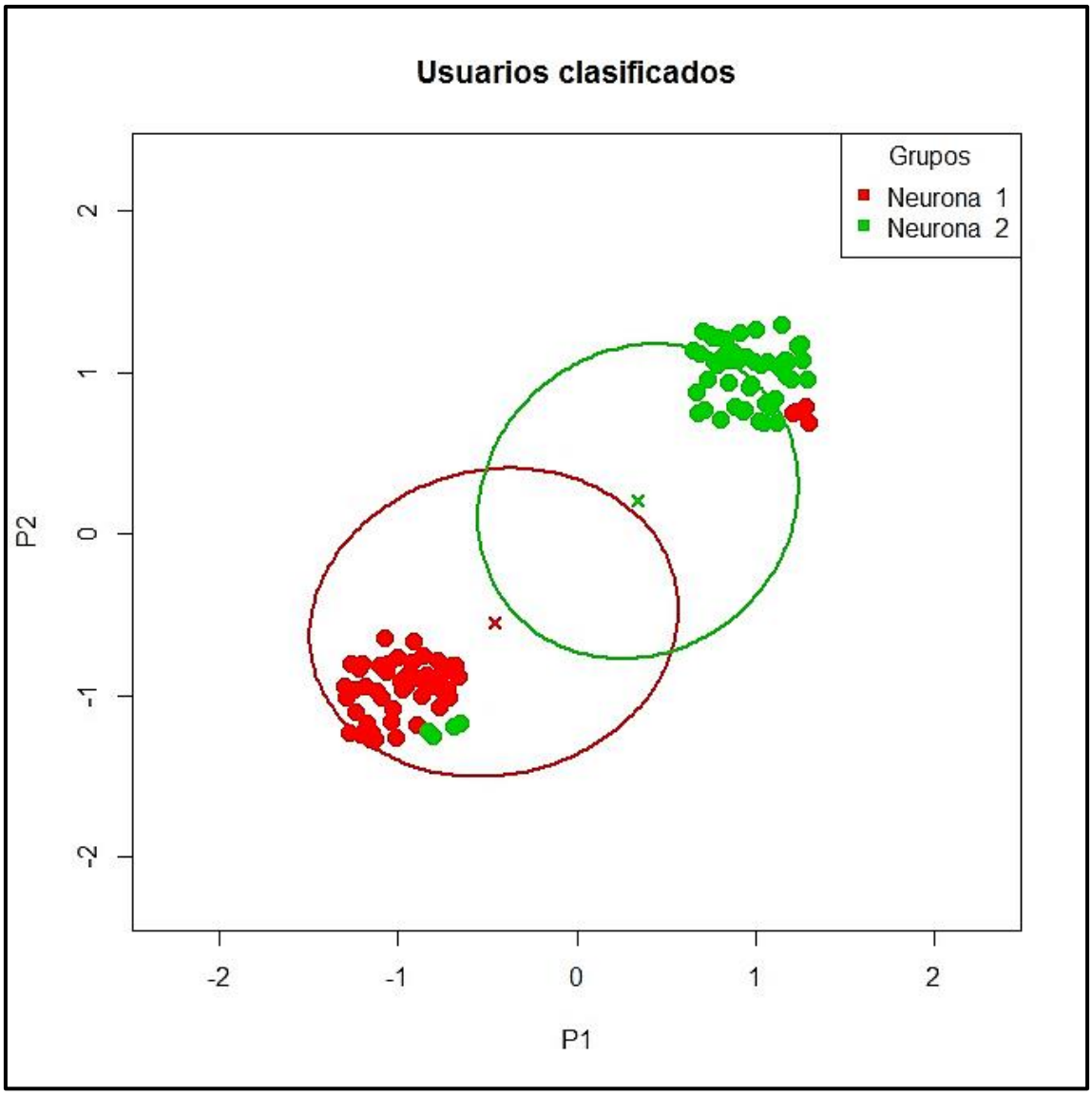


Ilustración 23. Gráfica de clasificación (datos 2, prueba 2)

En esta prueba se ha seleccionado una tasa de aprendizaje para la matriz de covarianza menor, así se le da más oportunidad de aprender a ambas neuronas puesto que se adaptan a los datos de entrada más lentamente. También se ha disminuido considerablemente la cantidad de varianza que describen las neuronas para forzar un mejor aprendizaje.

A pesar de que las neuronas han aprendido considerablemente bien se han realizado más pruebas con el objetivo de encontrar una configuración más óptima.

Prueba 3

En esta prueba y la siguiente (prueba 4) se obtuvieron resultados similares a los anteriores (ver ilustración 24). La configuración para esta red es:

- Número de épocas: 200
- Tasa de aprendizaje de la matriz de covarianza: 0.2
- Cantidad de varianza que describen las neuronas: 0.2
- Número de neuronas: 2

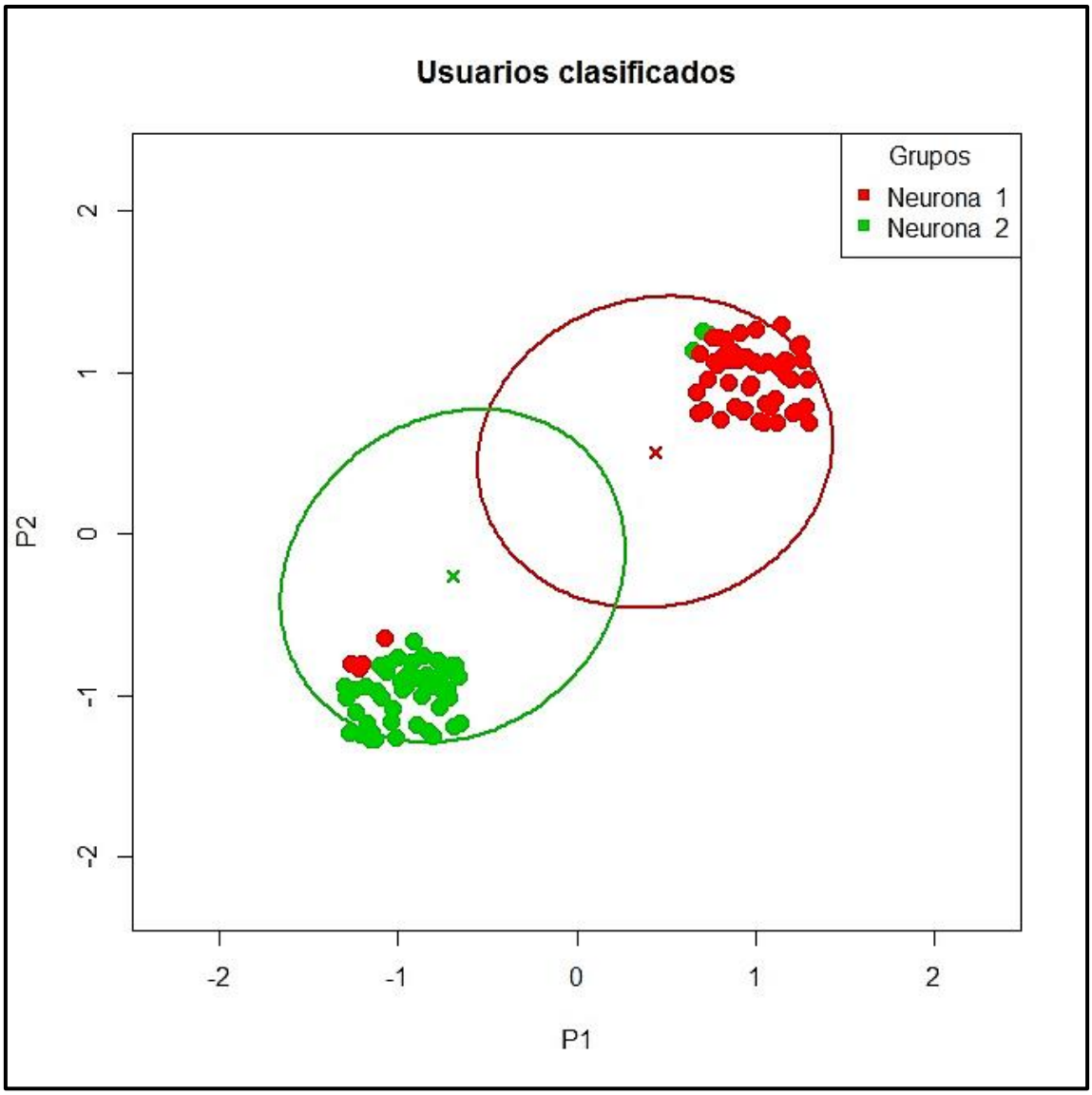


Ilustración 24. Gráfica de datos clasificados (datos 2, prueba 3)

Prueba 4

La configuración para esta prueba es:

- Número de épocas: 200
- Tasa de aprendizaje de la matriz de covarianza: 0.3
- Cantidad de varianza que describen las neuronas: 0.2
- Número de neuronas: 2

En la siguiente ilustración se puede observar el resultado de la clasificación (ver ilustración 25).

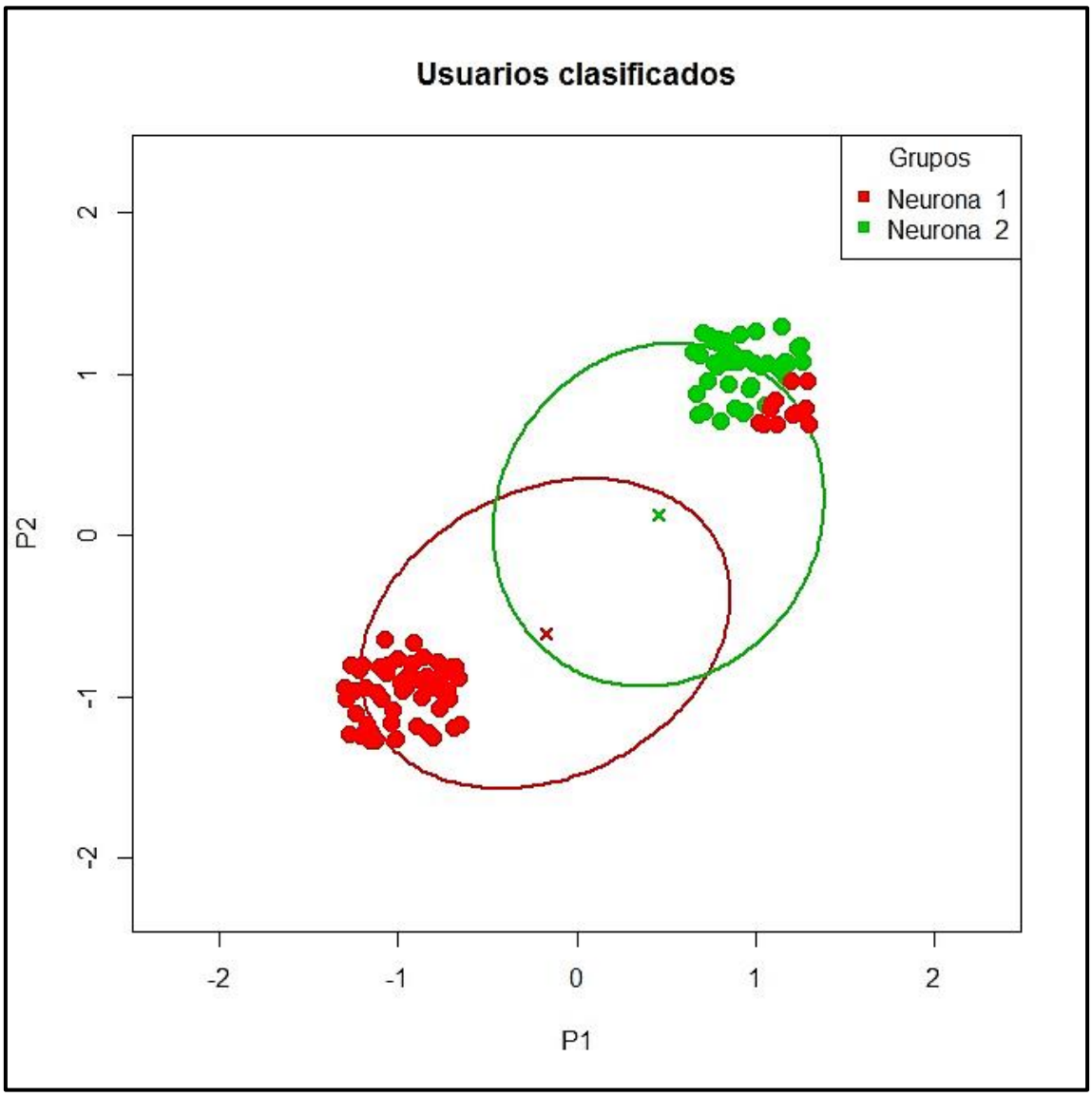


Ilustración 25. Gráfica de datos clasificados (datos 2, prueba 4)

Prueba 5

La configuración de esta prueba es la siguiente:

- Número de épocas: 100
- Tasa de aprendizaje de la matriz de covarianza: 0.4
- Cantidad de varianza que describen las neuronas: 0.2
- Número de neuronas: 2

El resultado de la clasificación de los datos tras el entrenamiento se puede observar en la siguiente ilustración (ver ilustración 26).

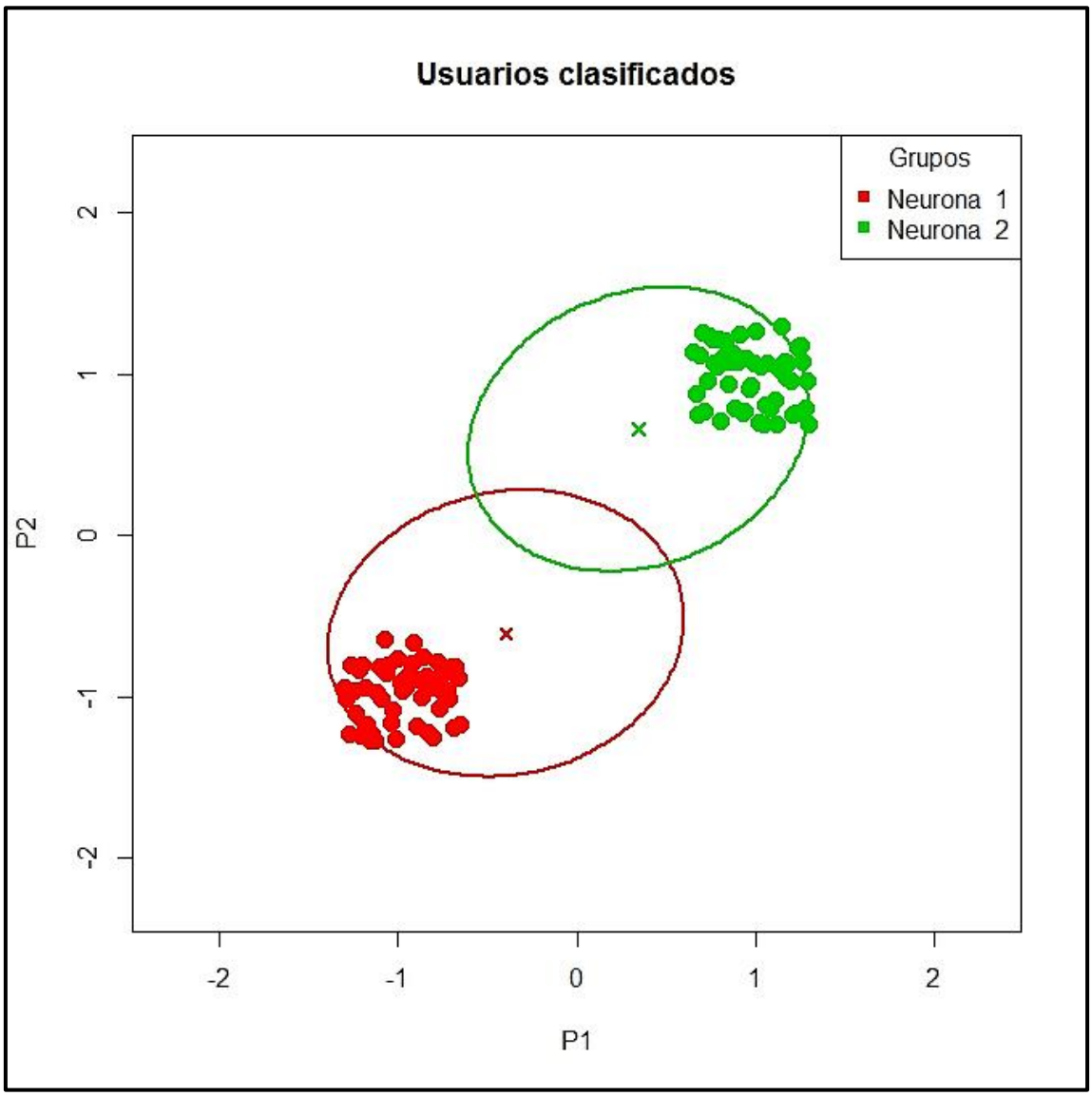


Ilustración 26. Gráfica de datos clasificados (datos 2, prueba 5)

En la ilustración anterior se puede observar que las neuronas han aprendido correctamente el patrón de los datos y tienden a acercarse a dichos datos y separarse entre ellas.

Capítulo IV. Diseño e implementación del sistema de recomendación

El sistema de recomendación se ha implementado, al igual que las pruebas, en dos scripts, *"Preparación y entorno de datos"* y *"Ejecutar entrenamiento y predicciones"*, definidos en el capítulo I. En el primer script (ver ilustración 2) se genera el conjunto de datos, posteriormente este conjunto se utiliza en el segundo script (ver ilustración 3), en el cual se entrena la red neurona y posteriormente se clasifica ese mismo conjunto de datos. Finalmente se obtiene la recomendación para cada usuario con la ayuda de la función *"recomendarProductos"* (ver ilustración 27).

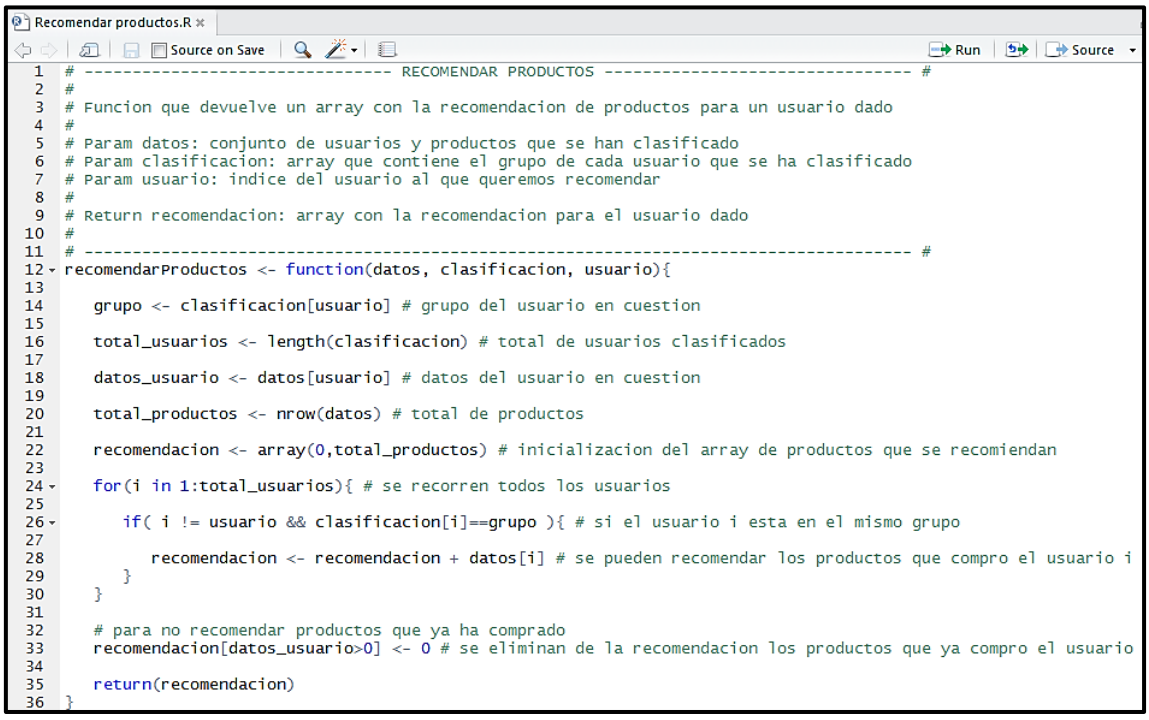


Ilustración 27. Recomendación de productos

Esta función recibe el conjunto de datos con los que se ha entrenado la red y la clasificación de esos mismos datos, además del índice del usuario al que se le desean recomendar productos. Para hallar la recomendación de productos primero se obtiene el grupo o neurona al que pertenece dicho usuario y sus datos de compra o valoraciones. Posteriormente se introducen los valores de las compras o valoraciones de otros usuarios clasificados en el mismo grupo y finalmente se retiran de este array los productos que el usuario ya había comprado o valorado para presentarle una buena recomendación.

Conclusiones

La realización de este trabajo de fin de grado ha supuesto un gran punto de partida en el mundo de las redes neuronales y las grandes posibilidades que ofrecen para desarrollar tecnologías con aplicación en todas las áreas que afecten a la sociedad o la economía en general.

Objetivos logrados

Al ver los resultados de las pruebas realizadas al sistema de recomendación basado en el modelo PCACL, se puede decir que el funcionamiento es correcto, y que el sistema de recomendación que es el objetivo de este TFG se ha logrado con una satisfacción muy positiva, ya que a nivel de la red neuronal se puede decir que aprende correctamente, y es una parte crucial en cualquier sistema basado sobre redes neuronales.

Desde una perspectiva de desarrollo personal, este trabajo ha puesto a prueba los conocimientos teóricos adquiridos en la carrera, especialmente en asignaturas relacionadas con la inteligencia artificial y estadística, además de la asignatura “Modelos de la computación” en la que se estudia, principalmente, las redes neuronales. También destaca el aprendizaje del lenguaje de programación R que ha sido un gran logro profesional y que da posibilidades de participar en proyectos relacionados con la inteligencia artificial y también en el área de Big Data.

Futuro del este trabajo

Este trabajo ha llevado a la práctica la implementado un sistema de recomendación basado en el modelo teórico PCACL y se ha desarrollado con datos ficticios. Se deberán continuar haciendo pruebas más exaustivas a la red y realizar entrenamientos y predicciones con datos reales para, en algún momento, poder llevar a la práctica este sistema e introducirlo en una aplicación de compra real.

Bibliografía

¿Qué es el análisis de componentes principales? (s.f.). Obtenido de Minitab: http://support.minitab.com/es-mx/minitab/17/topic-library/modeling-statistics/multivariate/principal-components-and-factor-analysis/what-is-pca/

Análisis de componentes principales. (s.f.). Obtenido de Wikipedia: https://es.wikipedia.org/wiki/An%C3%A1lisis_de_componentes_principales

Ballesteros, A. (s.f.). *Neural Network Framework*. Obtenido de http://www.redes-neuronales.com.es/tutorial-redes-neuronales/red-neuronal-competitiva-simple.htm

Carmona, F. (2007). *Curso básico de R.*

Filtrado colaborativo. (s.f.). Obtenido de Wikipedia: https://es.wikipedia.org/wiki/Filtrado_colaborativo

Jolliffe, I. T. (1986). *Principal component analysis.* Berlin: Springer-Verlag.

López Rubio, E., Ortiz de Lazcano Lobato, J. M., Muñoz Pérez, J., & Gómez Ruiz, J. A. (2004). Principal Components Analysis Competitive Learning, Neural Computation. 1-24.

R (lenguaje de programación). (s.f.). Obtenido de Wikipedia: https://es.wikipedia.org/wiki/R_(lenguaje_de_programaci%C3%B3n)

R Development Core Team. (2000). *Introducción a R.*

Sistema de recomendación. (s.f.). Obtenido de Wikipedia: https://es.wikipedia.org/wiki/Sistema_de_recomendaci%C3%B3n

Ilustraciones